

Трейси Пайлон

Дэн Пайлон

Включая
iOS7 и Xcode5

Программируем для iPhone и iPad

3-е издание

Освой
принципы
проектирования
интерфейсов
для iOS



Узнай все
о среде
разработки
Xcode



Создавай
приложения,
которые
приносят
деньги



Задействуй
мультимедийные
и GPS-функции
iPhone и iPad



Заходи
в App Store
и празднуй
победу

O'REILLY®

ПИТЕР®

Head First iPhone and iPad Development

Wouldn't it be dreamy if
there was a book to help me
learn how to develop iOS apps
that was more fun than going
to the dentist? It's probably
nothing but a fantasy...



3-rd Edition

Tracey Pilone
Dan Pilone

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Программируем для iPhone и iPad

Включая iOS7 и Xcode5

Хорошо бы найти книгу,
которая научила бы меня создавать
приложения для iOS и при этом была
приятнее визита к зубному врачу!
Как жаль, что это только мечты...

3-е издание

Трейси Пайлон

Дэн Пайлон



Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск

2014

Т. Пайлон, Д. Пайлон
Программируем для iPhone и iPad
3-е издание

Серия «Head First O'Reilly»

Перевел с английского *Е. Матвеев*

Заведующий редакцией	<i>П. Щеголев</i>
Руководитель проекта	<i>А. Юрченко</i>
Ведущий редактор	<i>Ю. Сергиенко</i>
Корректор	<i>С. Беляева</i>
Верстка	<i>Н. Лукьянова</i>

ББК 32.973.23-018

УДК 004.42

Пайлон Т., Пайлон Д.

П12 Программируем для iPhone и iPad. 3-е изд. — СПб.: Питер, 2014. — 336 с.: ил. — (Серия «Head First O'Reilly»).
ISBN 978-5-496-01083-2

С появлением iPhone мир изменился. Потом с появлением iPhone 4 он изменился снова. А теперь к iPhone добавился еще и революционный планшет iPad. Современные устройства на базе iOS используются в бизнесе и учебе, для работы и развлечений, и на App Store уже сейчас успешно работают десятки тысяч программистов и известных софтверных компаний.

Представим, что у вас появилась гениальная идея приложения для iPhone и iPad. С чего начать? Эта книга поможет вам разработать свое первое приложение в самые кратчайшие сроки. Вы не только узнаете, как спроектировать приложение для устройств Apple и сделать его уникальным, но и в совершенстве овладеете принципами программирования на Objective-C и инструментами iPhone SDK, в том числе Interface Builder и Xcode. Apple предоставляет программное обеспечение, эта книга дает знания — от вас потребуется лишь энтузиазм и желание научиться разрабатывать оригинальные и коммерчески успешные приложения для iPhone и iPad. Новое издание книги охватывает версии iOS7 и Xcode5.

Особенностью данного издания является уникальный способ подачи материала, выделяющий серию «Head First» издательства O'Reilly в ряду множества скучных книг, посвященных программированию.

6+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-1449316570 англ.

Authorized Russian translation of the English edition of Head First iPhone and iPad Development, 3rd Edition (ISBN 9781449316570) © 2013 Dan Pilone and Tracey Pilone. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

978-5-496-01083-2

© Перевод на русский язык ООО Издательство «Питер», 2014

© Издание на русском языке, оформление ООО Издательство «Питер», 2014

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Подписано в печать 27.05.14. Формат 84×108/16. Усл. п. л. 35,280. Тираж 1000. Заказ 0000.

Отпечатано в полном соответствии с качеством предоставленных издательством материалов
в Чеховский Печатный Двор. 142300, Чехов, Московская область, г. Чехов, ул. Полиграфистов, 1.

Посвящается всей моей семье: Чедвикам, Пайлонам и коллективу Element 84, которые поддерживали меня как дома, так и на работе. Также посвящается моему мужу, который всегда со мной и без которого моя жизнь не была бы такой интересной.

— **Трейси**

Посвящается моей жене, бизнес-партнеру и лучшему другу. Она вдохновила меня написать первое издание, а в этом выполнила большую часть работы. :)

— **Дэн**

Авторы

Трейси



Трейси Пайлон — соучредитель (вместе с Дэном Пайлоном) и руководитель проектов в Element 84 — начинающей фирме, занимающейся разработкой масштабируемых мобильных приложений и веб-приложений. Кроме непосредственного участия в разработке, она также работает с группами гибкой разработки Element 84 в области управления проектами и организации взаимодействия с клиентами.

До того как создать Element 84, Трейси несколько лет проработала в Вашингтоне руководителем строительных работ в области коммерческого строительства. Это ее четвертая книга из серии *Head First*; до этого вышли два издания этой книги и *Head First Algebra*.

Трейси получила диплом в области гражданского строительства в Политехническом институте Вирджинии, а также степень магистра педагогики в Университете Вирджинии; имеет разрешение на работу в качестве инженера-консультанта в штате Вирджиния. Вы можете читать ее сообщения в Twitter: @traceypilone.



ДЭН

Дэн Пайлон — основатель и партнер-распорядитель Element 84 — компании, занимающейся консультациями и разработкой мобильных приложений. Он проектировал и реализовывал системы для NASA, Hughes, ARINC, UPS и Naval Research Laboratory. В настоящее время руководит техническими проектами NASA, а также всеми проектами Element 84. Он часто выступает с докладами в сообществе, в последнее время ESIP, AGU и DC Ruby Users Group.

Дэн преподавал управление проектами, проектирование программного обеспечения и программирование в Католическом университете Вашингтона (федеральный округ Колумбия был инструктором на семинарах D.C. iPhone Bootcamp и написал несколько книг по разработке программного обеспечения, включая *Head First Software Development*, *UML 2.0 in a Nutshell* и *UML 2.0 Pocket Reference*. Вы можете читать его сообщения в Twitter: @danpilone.

Наши соавторы

Пол



Пол Пайлон — разработчик на платформе iOS и Rails из компании Element 84. Является автором приложений iHomework, а также приложений для iPhone, iPad и Mac для управления домашними заданиями. Пол писал программы для Naval Research Labs, Lockheed Martin, NASA и Cengage Learning. Вы можете читать его сообщения в Twitter: @paulpilone.

Бретт



Бретт Маклафлин — разработчик из компании Element 84; также занимался теорией познания. Рассматривает HTML5, CSS, JavaScript, Java и Rails не как скучные технологии и протоколы, а как возможность рассказывать интересные истории для пользователей. Также интересуется коммуникационными технологиями следующего поколения, от электронных книг до мобильных устройств. А в свободное время (да откуда оно возьмется?) Бретт обычно развлекается с видео, программирует и играет на гитаре... очень дорогой акустической гитаре.

Содержание (сводка)

	Введение	19
1	Первые шаги. <i>iOS и эпоха мобильности</i>	31
2	Основные паттерны iOS. <i>Строим приложение «с нуля»</i>	59
2.5	Интермедия. <i>Синтаксис</i>	105
3	Таблицы, представления и данные. <i>Таблица и представление</i>	119
4	Приложения с несколькими представлениями. <i>Такие важные подробности</i>	163
5	Процесс рецензирования, дизайн и устройства. <i>Как жить в «яблочном» мире</i>	199
6	Core Data и ячейки табличного представления. <i>Как поймать любимую передачу</i>	223
7	Реализация поиска с Core Data. <i>Поиск информации</i>	261
8	Core Data, map kit и Core location. <i>В поисках телефонной будки</i>	267

Содержание (настоящее)

Введение

Ваш мозг думает о разработке приложений для iOS.

Вы сидите за книгой и пытаетесь что-нибудь выучить, но ваш мозг считает, что вся эта писанина не нужна. Ваш мозг говорит: «Выгляни в окно! На свете есть более важные вещи, например сноуборд». Как заставить мозг думать, что ваша жизнь действительно зависит от умения разрабатывать приложения для iPhone и iPad?

Для кого написана эта книга?	20
Мы знаем, о чем вы думаете	21
Метапознание: наука о мышлении	23
Что можете сделать вы, чтобы заставить свой мозг повиноваться	25
Примите к сведению	26
Технические рецензенты	28
Благодарности	29

1 первые Шаги

iOS и эпоха мобильности

С появлением iPhone мир изменился. Когда Стив Джобс произнес эту фразу при выпуске iPhone, мало кто ему поверил. Прошло шесть лет, и устройства iPhone и iPad используются в бизнесе и медицине, а на платформе App Store работает множество программистов — от талантливых одиночек до известных фирм. Apple предоставляет программное обеспечение, мы поможем вам знаниями — от вас потребуется лишь энтузиазм.

Итак, вы хотите написать приложение для iOS...	32
Добро пожаловать во Вселенную Apple!	33
Приложения iOS пишутся на языке Objective-C	34
Все начинается с SDK	35
Это Сью, ваш новый начальник	36
Xcode и Git... неразлучные друзья	37
Xcode находится в центре любого проекта iOS	39
Эмулятор iOS	41
Программа хранится в файлах с исходным кодом	45
Редактор кода, центр управления... да еще и отладчик	50
Один iPhone, два iPhone, красный iPhone, синий iPhone...	52
Ваш инструментарий разработки для iPhone	58

2 основные паттерны iOS

Строим приложение «с нуля»

Итак, первое знакомство состоялось; теперь пришло время начать все заново. Вероятно, вы уже примерно представляете, с какими инструментами вам предстоит работать и как устроена среда Xcode. Пора изучить проблему более основательно и построить собственный проект. Как создать проект iOS, как связаны друг с другом компоненты приложений, на какие схемы взаимодействий следует ориентироваться? Вы это скоро узнаете — а пока переверните страницу...

Приложения iOS работают в полноэкранном режиме	60
Паттерн Модель-Представление-Контроллер	61
Начинаем с Xcode и Git	65
Переходим к построению интерфейса!	67
Переходим к построению интерфейса... дубль два	69
Xcode позволяет легко вносить косметические изменения	74
Элементы управления iOS изнутри	75
Действия создаются в редакторе графического интерфейса среды Xcode	77
Связывание элементов управления с действиями	79
И как добраться до текста?	88
Методы чтения и записи свойств создаются автоматически	89
Создайте свойство для текстового поля	91
Элементы управления связываются со ссылками IBOutlet	93
Простая работа с Twitter	96
Ваш инструментарий разработки для iOS	104

2.5

интерМедия

Синтаксис

Пора заняться мелочами. Мы написали пару приложений и более или менее разобрались с общей картиной. Пришло время досконально, построчно разбираться в мелочах. Почему в коде повсюду разбросаны символы @? Чем метод отличается от сообщения? Как именно работают свойства? Нас ждет краткая экскурсия по синтаксису Objective-C; потом можно будет вернуться к построению приложений.

Классы: интерфейс и реализация	106
Заголовочные файлы описывают интерфейс класса	107
Свойства используются для эффективности	110
Передача сообщений: как это происходит в Objective-C	113
Раз уж мы занялись сообщениями...	117
Ваш инструментарий синтаксиса	118

таблицы, представления и данные

3

Таблица и представление

В большинстве приложений iOS используется несколько представлений. Мы написали симпатичное приложение с одним представлением. И все же каждый, кто когда-либо пользовался смартфоном, знает, что типичное представление одним представлением не ограничивается. Самые впечатляющие приложения iOS используют при работе со сложной информацией несколько представлений. Мы начнем с навигационных контроллеров и табличных представлений наподобие тех, которые используются в приложениях Mail и Контакты. Вот только у нас они будут использоваться несколько иначе...

Поздравляем!	120
Приложение: каталог SpinCity	121
Как работают приложения iOS	122
Иерархические данные — за пределами табличного представления	124
А теперь нужно связать эти два представления...	127
Три представления в одном шаблоне	130
Паттерн MVC используется для разделения обязанностей...	135
Добавление нового класса	136
Свойства предоставляют доступ к атрибутам класса	138
Объекты доступа к данным скрывают низкоуровневые операции с данными	141
Поздравляем — вы только что создали свой первый объект DAO!	145
Таблица состоит из ячеек	151
Ваш инструментарий представлений	162

4

приложения с несколькими представлениями

Такие важные подробности

В большинстве приложений iOS используется более одного представления. Работа над приложением началась резво: мы воспользовались встроенными шаблонами и внесли довольно впечатляющие изменения в табличное представление. Пришло время заняться выводом подробной информации, подготовкой нового представления и управлением навигацией между ними. Многие популярные приложения в App Store предоставляют удобные и простые средства для работы с большими объемами данных. Приложение Spin City делает то же самое — оно помогает пользователю найти нужную пластинку, не перерывая многочисленные конверты на полках!

Детализированные представления	164
Табличные представления не всегда похожи на... таблицы	166
Замена UIViewController на UITableView Controller	168
Макет нового детализированного представления	170
Включение представления в раскладовку	171
Пути соединяют контроллеры представлений	175
Свяжите сцены в раскладовке	176
Пути позволяют подготовиться к отображению новой сцены	185
Обновите метод обратного вызова prepareForSegue	186
Для этого есть приложение список	189
Создайте новый список свойств	190
Мы должны загрузить каждый альбом из списка	192
Преобразование данных в список plist одной простой операцией	193
Ваш инструментарий представлений	197

5

процесс рецензирования, дизайн и устройства

Как жить в «яблочном» мире**Разработка iOS не ограничивается одним программированием.**

Наверняка вы уже слышали всевозможные ужасы. Процесс рецензирования Apple известен своей строгостью и обилием правил, которые должны соблюдать программисты. Да, не ждите, что все пойдет как по маслу, но если вы знаете, что делаете, — все отнюдь не так плохо. А кроме того, когда ваше приложение будет одобрено, перед вами откроется невероятно популярный магазин App Store... с множеством покупателей, готовых расстаться со своими долларами. Разве не заманчиво?

Это мир Apple... просто вы в нем живете	201
Проверка устройства — обязательная часть	207
Пример проверки устройства: камера	208
iOS берет на себя технические подробности	208
Хмм... устройство поддерживается, функция недоступна	209
Правила HIG помогают, а не мешают	211
Вы уже понемногу привыкаете к правилам HIG...	212
Дизайн = внешний вид + поведение	214
Пять главных отличий iOS 7	215
Информация к размышлению: iPad — не iPhone	216
Ваш инструментарий Apple	223

6

core data и ячейки табличного представления

Как поймать любимую передачу

Устраивайтесь поудобнее и послушайте историю — историю о судьбоносном путешествии. Сегодня часто возникает одна проблема: как работать с большими объемами данных и организовать их представление в формате, подходящем для мобильных устройств? Это можно делать многими способами, включая обработку данных и представление в формате, удобном для навигации и интерпретации. Для примера возьмем программу телепередач: в эфире идет великое множество программ. Что делать поклоннику «Острова Гиллигана»?

Это ваше приложение	226
Это ваше приложение с данными	227
Знакомство с Core Data	228
...Кстати, о данных	229
HFN	229
Приложение Gilligizer	230
Core Data начинается с данных	232
Core Data работает с сущностями	235
Для описания сущностей Core Data использует модель управляемых объектов	236
Построение сущности Show	237
У нас есть объект... теперь его необходимо вывести	252
Вывод сущности в приложении Gilligizer	253
Ваш инструментарий Core Data	260

1

реализация поиска с core data

Поиск информации

Просто просматривать данные уже недостаточно. Наступила эпоха больших данных, и с обычным просмотром далеко не уйдешь. Вероятно, объем данных на вашем телефоне еще не измеряется в петабайтах, и все же он достаточно велик, чтобы функции сортировки и фильтрации упростили работу с ними. Core Data содержит встроенные средства покорения больших объемов данных, и в этой главе вы научитесь пользоваться ими!

Приложение работает, но его возможности ограничены...	262
Используйте NSFetchRequest для описания поиска	267
Давайте попробуем...	267
В iOS 7 поддержка поиска встроена в Core Data и UIKit	270
Использование предикаторов для фильтрации данных	271
Предикат NSFetchRequest определяет возвращаемые данные	272
Это был вопрос с подвохом...	274
Ваш инструментарий поиска	281



core data, map kit и core location

В поисках телефонной будки

Пришло время заняться расширенными возможностями.

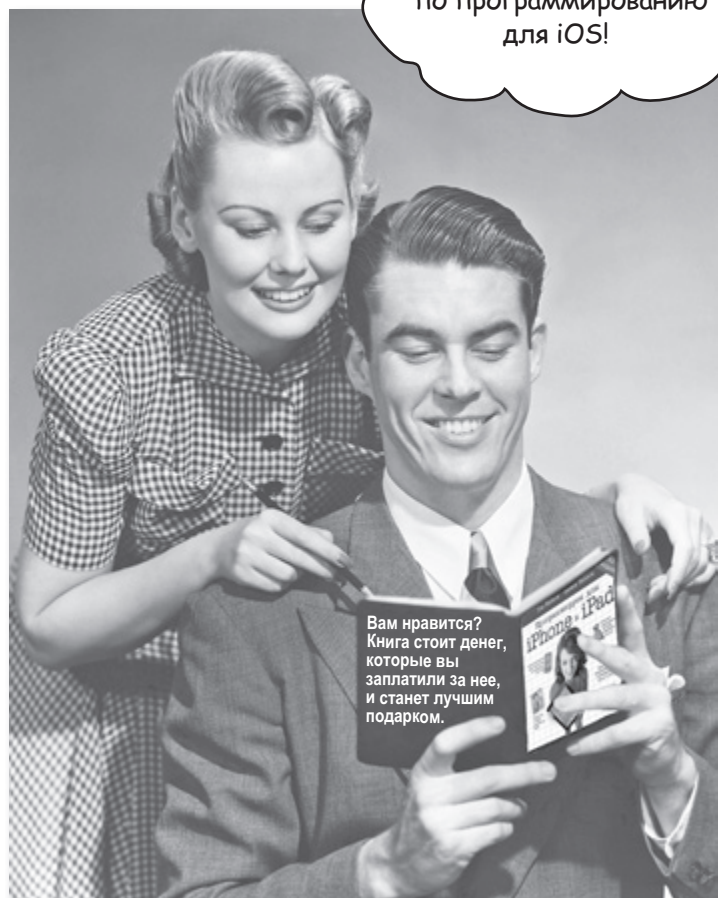
Устройства на базе iOS содержат массу полезных встроенных возможностей. И iPhone, и iPad — это одновременно и компьютер, и библиотека, и фотоаппарат, и видеокамера, и GPS-навигатор. Область технологий геопозиционирования еще только развивается, но уже сейчас она предоставляет много полезных возможностей. К счастью, iOS позволяет относительно легко использовать эти функции оборудования.

Все старое становится новым — клевым	284
Приложение, iPad и телефонные будки	285
Приложения iOS предназначены только для чтения (в какой-то степени...)	295
Структура приложения iOS определяет, где можно читать и записывать данные	296
Знакомьтесь... UIImagePickerController	297
Использование карты действий	304
Где эта улица, где этот дом?	311
Core Location может определить текущее местоположение несколькими способами	312
Map Kit поддерживается всеми устройствами iOS	323
Аннотации: придется поработать	330
Полная реализация протокола аннотаций	331
Ваш инструментарий	334

КАК ПОЛЬЗОВАТЬСЯ ЭТОЙ КНИГОЙ

Введение

Не могу поверить, что они
включили **такое** в книгу
по программированию
для iOS!



В этом разделе мы ответим на насущный
вопрос: «Так почему они включили **ТАКОЕ**
в книгу по программированию для iOS?»

Для кого написана эта книга?

Если вы ответите «да» на все следующие вопросы...

- 1 У вас уже есть опыт программирования?
- 2 Вы хотите **изучить, запомнить, понять и научиться разрабатывать** приложения для iOS, чтобы писать собственные приложения и распространять их через App Store?
- 3 Вы предпочитаете **оживленную беседу сухим, скучным академическим лекциям?**

...то эта книга для вас.

Полезно иметь некоторое представление об объектно-ориентированном программировании. Опыт программирования для Mac пригодится, но не является обязательным.

Кому эта книга не подойдет?

Если вы ответите «да» на любой из следующих вопросов:

- 1 Вы **абсолютно не разбираетесь** в программировании?
- 2 Вы уже занимаетесь разработкой приложений для iOS и ищете **справочник** по Objective-C?
- 3 Вы **боитесь попробовать что-нибудь новое?** Скорее пойдете к зубному врачу, чем наденете полосатое с клетчатым? Считаете, что техническая книга с телефонной будкой серьезной быть не может?

...эта книга не для вас.

Найдите книгу Head First Programming, в ней приведено отличное введение в объектно-ориентированную разработку. А потом возвращайтесь к нам.

[Заметка от отдела продаж: вообще-то эта книга для любого, у кого есть деньги.]



Мы знаем, о чем вы думаете.

«Разве серьезные книги по программированию *такие?*»

«И почему здесь столько рисунков?»

«Можно ли так чему-нибудь *научиться?*»

Мы знаем, о чем думает ваш мозг

Мозг жаждет новых впечатлений. Он постоянно ищет, анализирует, *ожидает* чего-то необычного. Он так устроен, и это помогает нам выжить.

Как наш мозг поступает со всеми обычными, повседневными вещами? Он всеми силами пытается оградиться от них, чтобы они не мешали его *настоящей* работе — сохранению того, что действительно *важно*. Мозг не считает нужным сохранять скучную информацию. Она не проходит фильтр, отсекающий «очевидно несущественное».

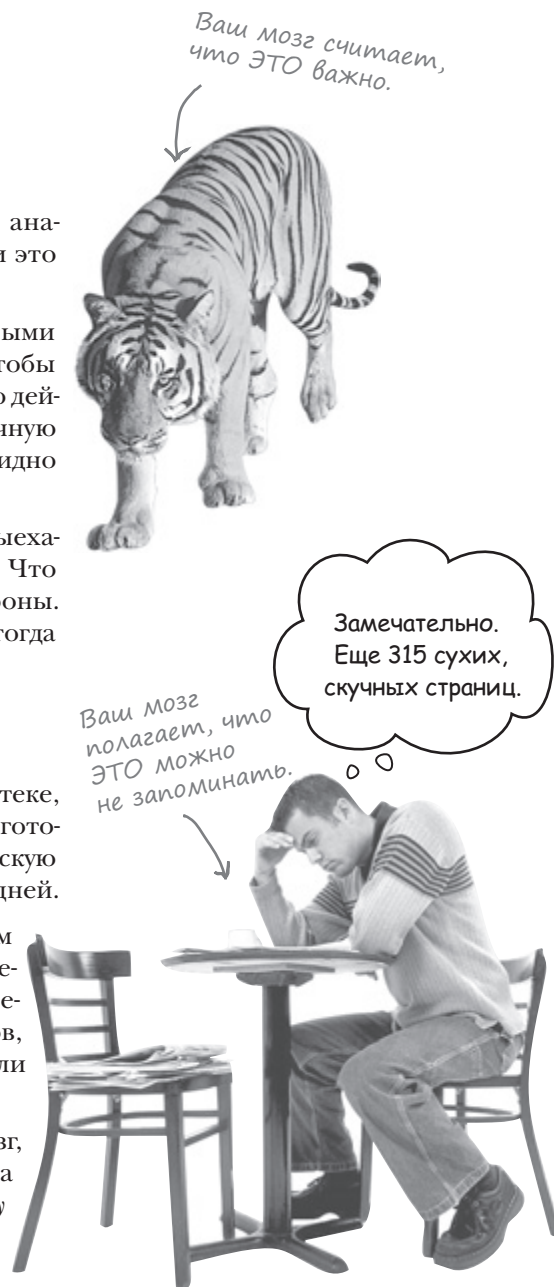
Но как же мозг *узнает*, что важно? Представьте, что вы выехали на прогулку и вдруг прямо перед вами появляется тигр. Что происходит в вашей голове и теле? Активизируются нейроны. Вспыхивают эмоции. Происходят химические реакции. И тогда ваш мозг понимает...

Конечно, это важно! Не забывать!

А теперь представьте, что вы находитесь дома или в библиотеке, в теплом, уютном месте, где тигры не водятся. Вы учитесь — готовитесь к экзамену. Или пытаетесь освоить сложную техническую тему, на которую вам выделили неделю... максимум десять дней.

И тут возникает проблема: ваш мозг пытается оказать вам услугу. Он старается сделать так, чтобы на эту *очевидно* несущественную информацию не тратились драгоценные ресурсы. Их лучше потратить на что-нибудь важное. На тигров, например. Или на то, что к огню лучше не прикасаться. Или что на лыжах не стоит кататься в футболке и шортах.

Нет простого способа сказать своему мозгу: «Послушай, мозг, я тебе, конечно, благодарен, но какой бы скучной ни была эта книга и пусть мой датчик эмоций сейчас на нуле, я *хочу* запомнить то, что здесь написано».



Эта книга для тех, кто хочет учиться.

Как мы что-то узнаем? Сначала нужно это «что-то» понять, а потом не забыть. Затолкать в голову побольше фактов недостаточно. Согласно новейшим исследованиям в области когнитивистики, нейробиологии и психологии обучения, для усвоения материала требуется что-то большее, чем простой текст на странице. Мы знаем, как заставить ваш мозг работать.

Основные принципы серии Head First:

Наглядность. Графика запоминается лучше, чем обычный текст, и значительно повышает эффективность восприятия информации (до 89 % — по данным исследований). Кроме того, материал становится более понятным. Текст размещается на рисунках, к которым он относится, а не под ними или на соседней странице.

Разговорный стиль изложения. Недавние исследования показали, что при разговорном стиле изложения материала (вместо формальных лекций) улучшение результатов на итоговом тестировании достигает 40 %. Рассказывайте историю, вместо того чтобы читать лекцию. Не относитесь к себе слишком серьезно. Что привлечет ваше внимание: занимательная беседа за столом или лекция?

Активное участие читателя. Пока вы не начнете напрягать извилины, в вашей голове ничего не произойдет. Читатель должен быть заинтересован в результате; он должен решать задачи, формулировать выводы и овладевать новыми знаниями. А для этого необходимы упражнения и каверзные вопросы, в решении которых задействованы оба полушария мозга и разные чувства.

Погодите... Вы же обещали объяснить, что там происходит с выборкой и контроллерами...



Все это, конечно, хорошо, но где информация об альбоме?



Привлечение — и сохранение — внимания читателя.

Ситуация, знакомая каждому: «Я очень хочу изучить это, но засыпаю на первой странице». Мозг обращает внимание на интересное, странное, притягательное, неожиданное. Изучение сложной технической темы не обязано быть скучным. Интересное узнается намного быстрее.

Обращение к эмоциям. Известно, что наша способность запоминать в значительной мере зависит от эмоционального сопереживания. Мы запоминаем то, что нам небезразлично. Мы запоминаем, когда что-то чувствуем. Нет, сентименты здесь ни при чем: речь идет о таких эмоциях, как удивление, любопытство, интерес и чувство «Да я крут!» при решении задачи, которую окружающие считают сложной, — или когда вы понимаете, что разбираетесь в теме лучше, чем всезнайка Боб из технического отдела.

Метапознание: наука о мышлении

Если вы действительно хотите быстрее и глубже усваивать новые знания — задумайтесь над тем, как вы задумываетесь. Учитесь учиться.

Мало кто из нас изучает теорию метапознания во время учебы. Нам положено учиться, но нас редко этому *учат*.

Но раз вы читаете эту книгу, то, вероятно, хотите освоить программирование для iOS, и по возможности быстрее. Вы хотите *запомнить* прочитанное, а для этого абсолютно необходимо сначала *понять* прочитанное.

Чтобы извлечь максимум пользы из учебного процесса, нужно заставить ваш мозг воспринимать новый материал как Нечто Важное. Критичное для вашего существования. Такое же важное, как тигр. Иначе вам предстоит бесконечная борьба с вашим мозгом, который всеми силами уклоняется от запоминания новой информации.

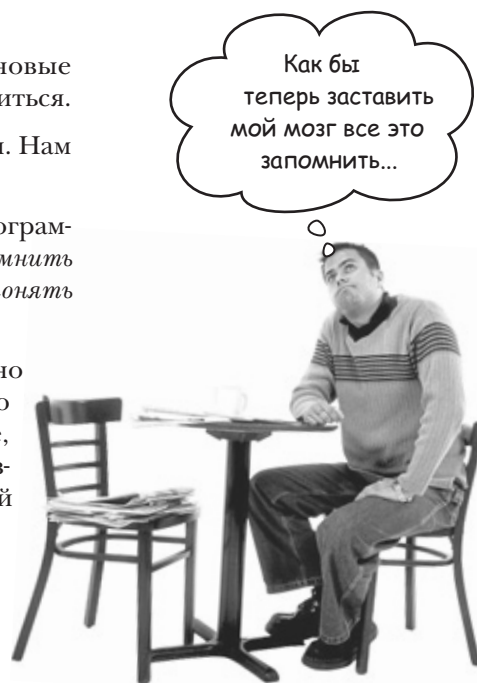
Как же УБЕДИТЬ мозг, что программирование для iOS не менее важно, чем тигр?

Есть способ медленный и скучный, а есть быстрый и эффективный. Первый основан на тупом повторении. Всем известно, что даже самую скучную информацию *можно* запомнить, если повторять ее снова и снова. При достаточном количестве повторений ваш мозг прикидывает: «*Вроде бы* несущественно, но раз одно и то же повторяется *столько раз...* Ладно, уговорил».

Быстрый способ основан на **повышении активности мозга** и особенно на сочетании разных ее *видов*. Доказано, что все факторы, перечисленные на предыдущей странице, помогают вашему мозгу работать на вас. Например, исследования показали, что размещение слов *внутри* рисунков (а не в подписях, в основном тексте и т. д.) заставляет мозг анализировать связи между текстом и графикой, а это приводит к активизации большего количества нейронов. Больше нейронов — выше вероятность того, что информация будет сочтена важной и достойной запоминания.

Разговорный стиль тоже важен: обычно люди проявляют больше внимания, когда они участвуют в разговоре, так как им приходится следить за ходом беседы и высказывать свое мнение. Причем мозг совершенно не интересуется, что вы «разговариваете» с книгой! С другой стороны, если текст сух и формален, то мозг чувствует то же, что чувствуете вы на скучной лекции в роли пассивного участника. Его клонит в сон.

Но рисунки и разговорный стиль — это только начало.



Вот что сделали Мы:

Мы использовали **рисунки**, потому что мозг лучше приспособлен для восприятия графики, чем текста. С точки зрения мозга рисунок стоит 1024 слова. А когда текст комбинируется с графикой, мы внедряем текст прямо в рисунки, потому что мозг при этом работает эффективнее.

Мы используем **избыточность**: повторяем одно и то же несколько раз, применяя разные средства передачи информации, обращаемся к разным чувствам — и все для повышения вероятности того, что материал будет закодирован в нескольких областях вашего мозга.

Мы используем концепции и рисунки несколько **неожиданным** образом, потому что мозг лучше воспринимает новую информацию. Кроме того, рисунки и идеи обычно имеют *эмоциональное содержание*, потому что мозг обращает внимание на биохимию эмоций. То, что заставляет нас *чувствовать*, лучше запоминается — будь то *шутка*, *удивление* или *интерес*.

Мы используем *разговорный стиль*, потому что мозг лучше воспринимает информацию, когда вы участвуете в разговоре, а не пассивно слушаете лекцию. Это происходит и при *чтении*.

В книгу включены многочисленные упражнения, потому что мозг лучше запоминает, когда вы что-то делаете. Мы постарались сделать их непростыми, но интересными — то, что предпочитает большинство читателей.

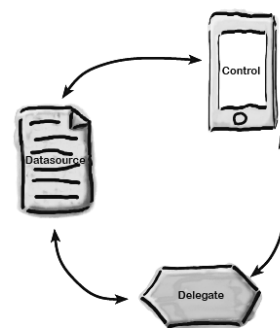
Мы совместили *несколько стилей обучения*, потому что одни читатели предпочитают пошаговые описания, другие стремятся сначала представить «общую картину», а третьим хватает фрагмента кода. Независимо от ваших личных предпочтений полезно видеть несколько вариантов представления одного материала.

Мы постарались задействовать *оба полушария вашего мозга*; это повышает вероятность усвоения материала. Пока одна сторона мозга работает, другая часто имеет возможность отдохнуть; это повышает эффективность обучения в течение продолжительного времени.

А еще в книгу включены *истории* и упражнения, отражающие другие точки зрения. Мозг глубже усваивает информацию, когда ему приходится оценивать и выносить суждения.

В книге часто встречаются **вопросы**, на которые не всегда можно дать простой ответ, потому что мозг быстрее учится и запоминает, когда ему приходится что-то делать. Невозможно накачать *мышцы*, наблюдая за тем, как занимаются *другие*. Однако мы позаботились о том, чтобы усилия читателей были приложены в верном направлении. Вам не придется ломать голову над невразумительными примерами или разбираться в сложном, перенасыщенном техническим жаргоном или слишком лаконичном тексте.

В историях, примерах, на картинках используются *люди* — потому что вы тоже человек. И ваш мозг обращает на людей больше внимания, чем на *неодушевленные предметы*.

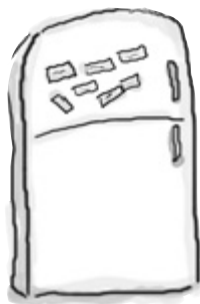


**КЛЮЧЕВЫЕ
МОМЕНТЫ**



Беседа у камина





*Вырежьте и прикрепите
на холодильник.*

Что можете сделать Вы, чтобы заставить мозг повиноваться

Мы свое дело сделали. Остальное за вами. Эти советы станут отправной точкой; прислушайтесь к своему мозгу и определите, что вам подходит, а что не подходит. Пробуйте новое.

- 1 Не торопитесь. Чем больше вы поймете, тем меньше придется запоминать.
- 6 Пейте воду. И побольше.

Просто читать недостаточно. Когда книга задает вам вопрос, не переходите к ответу. Представьте, что кто-то *действительно* задает вам вопрос. Чем глубже ваш мозг будет мыслить, тем скорее вы поймете и запомните материал.

Мозг лучше всего работает в условиях высокой влажности. Дегидратация (которая может наступить еще до того, как вы почувствуете жажду) снижает когнитивные функции.

- 2 Выполняйте упражнения, делайте заметки.
- 7 Прислушивайтесь к своему мозгу.

Мы включили упражнения в книгу, но выполнять их за вас не собираемся. И не *разглядывайте* упражнения. **Берите карандаш и пишите.** Физические действия *во время* учения повышают его эффективность.

Следите за тем, когда мозг начинает уставать. Если вы начинаете поверхностно воспринимать материал или забываете только что прочитанное, пора сделать перерыв.

- 3 Читайте врезки.
- 8 Чувствуйте!

Это значит: читайте всё. **Врезки — часть основного материала!** Не пропускайте их.

Ваш мозг должен знать, что материал книги действительно *важен*. Переживайте за героев наших историй. Придумывайте подписи к фотографиям. Поморщиться над неудачной шуткой лучше, чем не почувствовать ничего.

- 4 Не читайте другие книги после этой перед сном.
- 9 Пишите побольше кода!

Часть обучения (особенно перенос информации в долгосрочную память) происходит после того, как вы откладываете книгу. Ваш мозг не сразу усваивает информацию. Если во время обработки поступит новая информация, часть того, что вы узнали ранее, может быть потеряна.

Есть только один способ научиться программировать: **пишите код**. Именно этим мы и будем заниматься в книге. Программирование — это искусство, и чтобы овладеть им, необходимо тренироваться. Мы вам поможем: в каждой главе есть упражнения для читателя. Не пропускайте их; во время работы над упражнениями происходят очень важные вещи. Мы приводим решения для всех упражнений — не бойтесь **заглянуть в решение**, если окажетесь в тупике! Но обязательно постарайтесь решить задачу, прежде чем обращаться к решению. И обязательно добейтесь того, чтобы решение заработало, прежде чем переходить к следующей части книги.

- 5 Говорите вслух.

Речь активизирует другие участки мозга. Если вы пытаетесь что-то понять или лучше запомнить, произнесите вслух. А еще лучше — попробуйте объяснить кому-нибудь другому. Вы будете быстрее усваивать материал и, возможно, откроете для себя что-то новое.

Примите к сведению

Это учебник, а не справочник. Мы намеренно убрали из книги все, что могло бы помешать изучению материала, над которым вы работаете. И при первом чтении книги начинать следует с самого начала, потому что книга предполагает наличие у читателя определенных знаний и опыта.

Мы начнем с готового приложения, код которого загружается из GitHub.

Хотя эта книга посвящена программированию для iOS, мы также хотим научить вас пользоваться некоторыми инструментами, относящимися к разработке в целом. Итак, чтобы упростить первые шаги, мы предоставим вам полный код приложения, который нужно будет немного изменить (а не создавать «с нуля»).

Процесс отправки приложений в книге не рассматривается.

А в предыдущих изданиях рассматривался. Но есть два обстоятельства, которые усложняют рассмотрение этой темы: во-первых, после регистрации в программе Apple Developer Program некоторые сведения подпадают под действие соглашения о неразглашении информации, а во-вторых, разработка iOS со временем усложняется. В этой книге мы направили усилия на то, чтобы снабдить вас необходимой информацией. Сейчас для того, чтобы подготовить приложение к отправке, придется поработать больше, чем в прежние времена.

Основное внимание уделяется функциональности, которую можно протестировать в эмуляторе.

В iOS SDK входит замечательный (и к тому же бесплатный!) инструмент для тестирования приложений на компьютере. Эмулятор позволяет проверить работу кода без установки его на реальном устройстве или отправки в App Store. Однако возможности эмулятора не безграничны. Некоторые нетривиальные функции iOS невозможно протестировать в эмуляторе — например, акселерометр и компас. Соответственно, в книге эти возможности подробно не рассматриваются, потому что мы хотим научить вас быстро и легко создавать приложения.

Упражнения ОБЯЗАТЕЛЬНЫ.

Упражнения являются частью основного материала книги. Одни упражнения способствуют запоминанию материала, другие помогают лучше понять его, третьи ориентированы на его практическое применение. **Не пропускайте упражнения.**

Повторение применяется намеренно.

У книг этой серии есть одна принципиальная особенность: мы хотим, чтобы вы действительно хорошо усвоили материал. И чтобы вы запомнили все, что узнали. Большинство справочников не ставят своей целью успешное запоминание, но это не справочник, а учебник, поэтому некоторые концепции излагаются в книге по несколько раз.

Мы постарались сделать примеры по возможности компактными.

Нашим читателям не нравится пробираться через 200 строк примера в поисках двух строк, которые необходимо понять. Большинство примеров в книге приводится в минимальном контексте, чтобы часть, которую вы изучаете, была простой и понятной. Не ждите, что все примеры будут идеально надежными и даже законченными — они написаны для обучения и не всегда обладают полноценной работоспособностью.

Мы разместили код в репозитории GitHub, чтобы при необходимости вы могли скопировать все приложения и весь код, входящий в них. Код доступен по адресу:

<https://github.com/dpilone/Head-First-iPhone-iPad-Development-3rd-Edition>

Упражнения «Мозговой штурм» не имеют ответов.

В некоторых из них правильного ответа вообще нет, в других вы должны сами решить, насколько правильны ваши ответы (это является частью процесса обучения). В некоторых упражнениях приводятся подсказки, которые помогут вам найти нужное направление.

Системные требования

Чтобы программировать для iPhone и iPad, вам понадобится Mac на базе Intel. При написании книги мы использовали OS X версии 10.8.5 и Xcode 5.0. Впрочем, если вы работаете со старой версией Xcode, принципиальных отличий не так уж много. Для тестирования расширенных возможностей — акселерометра, камеры и т. д. — вам понадобится реальное устройство iPhone, iPod Touch или iPad и сертификат зарегистрированного разработчика. В главе 1 мы расскажем, где взять SDK и документацию Apple, так что пока об этом можно не беспокоиться.

Научные рецензенты

Майкл Моррисон



Джо Хек



Шон Мерфи



Рене Янссен Моррисон



Роберто Луис



Эрик Шеферд



Рич Розен



Рич Розен — один из соавторов *Mac OS X for Unix Geeks*. Также совместно с Леоном Шкларом (Leon Shklar) работал над *Web Application Architecture: Principles, Protocols & Practices* — учебником по разработке нетривиальных веб-приложений.

Шон Мерфи стал поклонником Сосоа уже более 10 лет назад. Он участвует в проектах с открытым кодом (таких, как Camino) и работает независимым проектировщиком и разработчиком приложений для iOS.

Джо Хек — программист, технический директор, автор и преподаватель с более чем 25-летним опытом работы с компьютерами. Он пишет приложения для платформы iPhone, начиная с первой бета-версии. Джо — основатель Seattle Xcoders, а также автор SeattleBus — приложения для iPhone, в реальном времени предоставляющего информацию о движении общественного транспорта в Сиэтле.

Эрик Шеферд начал программировать в 9-летнем возрасте, и с тех пор не сворачивал с выбранного пути. Он писал документацию для разработчиков с 1997 года, а в настоящее время руководит подготовкой документации в Mozilla.

Майкл Моррисон — писатель, разработчик и автор книг *Head First JavaScript*, *Head First PHP & MySQL* и даже нескольких «серьезных» книг, в которых нет стрелочек, человечков и магнитов. Майкл — основатель Stalefish Labs (www.stalefishlabs.com), развлекательно-образовательной компании, специализирующейся на играх и интерактивных программах, включая приложения для iPhone.

Роберто Луис — молодой программист, которому нравится изучать новые языки и средства разработки. Он получил степень по информатике в университете Autonoma de Madrid (Испания), и на протяжении своей карьеры участвовал в разработке для настольных, мобильных и облачных систем.

Рене Янссен — мультимедиадизайнер из Нидерландов, владелец компании Ducklord Studio. Он начал свою карьеру как дизайнер, работал в Indesign, Photoshop и Illustrator, но решил узнать больше и перешел к изучению языков — от HTML, CSS, PHP и MySQL до Actionscript. Он несколько лет проработал на Flash и Actionscript, после чего перешел к разработке iOS, Xcode и Objective-C.

Благодарности

Нашему редактору:

Мы благодарны Кортни Нэш, которая работала над всеми тремя изданиями этой книги (с 2009 года!), от начала и до конца. Нам пришлось искать обходные пути для решения проблем циклов выпуска Apple и это было непросто — но благодаря Кортни у нас все-таки получилось!

Кортни Нэш



Группе O'Reilly:

Спасибо всем талантливым специалистам из O'Reilly, которые доводили до ума наши файлы и всегда оставались «на связи», когда нам требовалась помощь.

Нашему практиканту:

Спасибо Джаянту Пратхипати — первому практиканту Element 84, который вызвался помочь нам со снимками экранов, которые необходимо было обновить для iOS7.

Нашим друзьям и семье:

Спасибо всем Пайлонам и Чедвикам, которые помогали нам с детьми и с пониманием относились к тому, что мы в неподходящие моменты усаживались за компьютеры.

Спасибо всем нашим друзьям из Element 84, которые нам помогали и делились своим мнением.

Спасибо Полу Пайлону, который помог нам в написании кода для книги и достаточно быстро справился с обновлением материала для iOS7.

Спасибо Бретту Маклафлину, который работал рядом с нами и предоставил нам другого специалиста, знакомого с тонкостями InDesign.

Спасибо Винни и Нику, которые во время работы над книгой ходили на тренировки по баскетболу и тэквондо. Надеемся, они уже смогут помочь нам в работе над следующим изданием!

1 Первые шаги

iOS и эпоха мобильности

Не понимаю, что все так носятся с iPhone и iPad? Мой телефон работает не хуже, а записывать можно и на бумаге!



С появлением iPhone мир изменился. Когда Стив Джобс произнес эту фразу при выпуске iPhone, мало кто ему поверил. Прошло шесть лет, и устройства iPhone и iPad используются в бизнесе и медицине, а на платформе App Store работает множество программистов — от талантливых одиночек до известных фирм. Apple предоставляет программное обеспечение, мы поможем вам знаниями — от вас потребуются лишь энтузиазм.

Итак, вы хотите написать приложение для iOS...

Когда-то разработка мобильных приложений считалась экзотикой, одним из самых передовых (и даже модных) направлений. Эти времена прошли. Мобильные приложения получили повсеместное распространение. Сейчас в мире насчитывается более 500 миллионов устройств iOS. Создавая приложение для iOS, вы получаете миллионы потенциальных пользователей (пусть даже довольно требовательных).

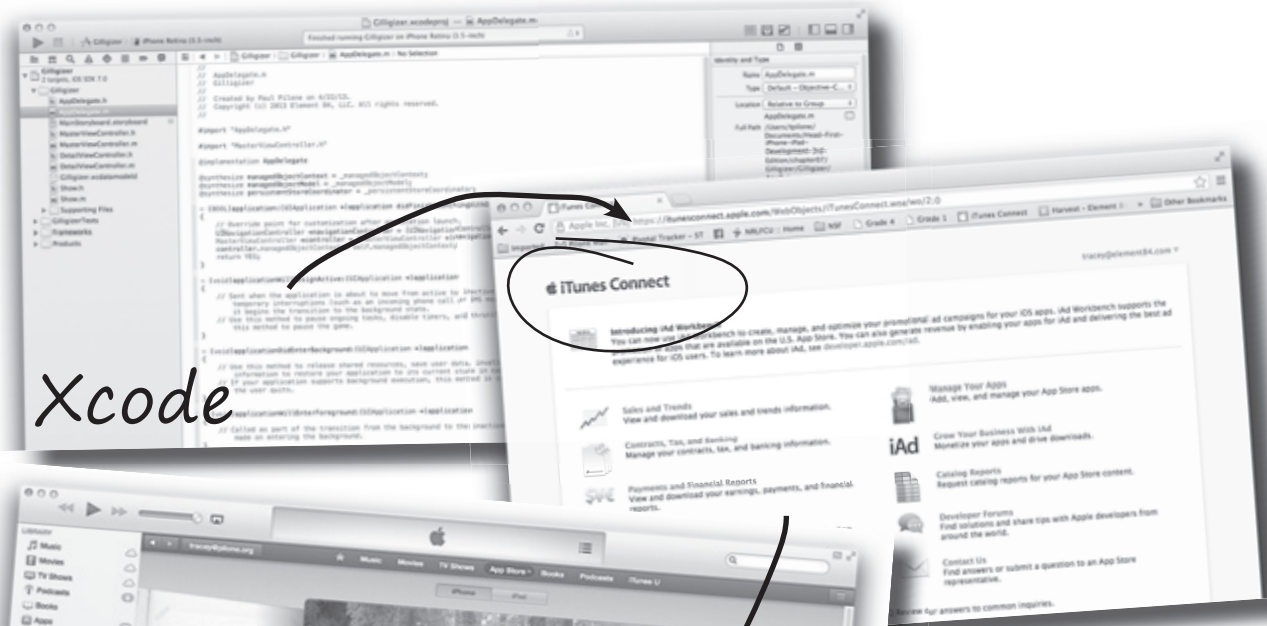
На заре эпохи мобильной разработки можно было обойтись простым «фонариком» или приложением, издававшим звуки при нажатии кнопки. Сегодня пользователи ждут от вас большего. Они хотят видеть графику высокого разрешения, поддержку разных вариантов ориентации устройства и быстрые, надежные приложения. А теперь хорошая новость: беритесь за книгу, изучайте примеры, пишите свой код — и ваше имя появится в iTunes App Store рядом с названиями таких компаний, как EA Games, и самой фирмы Apple!



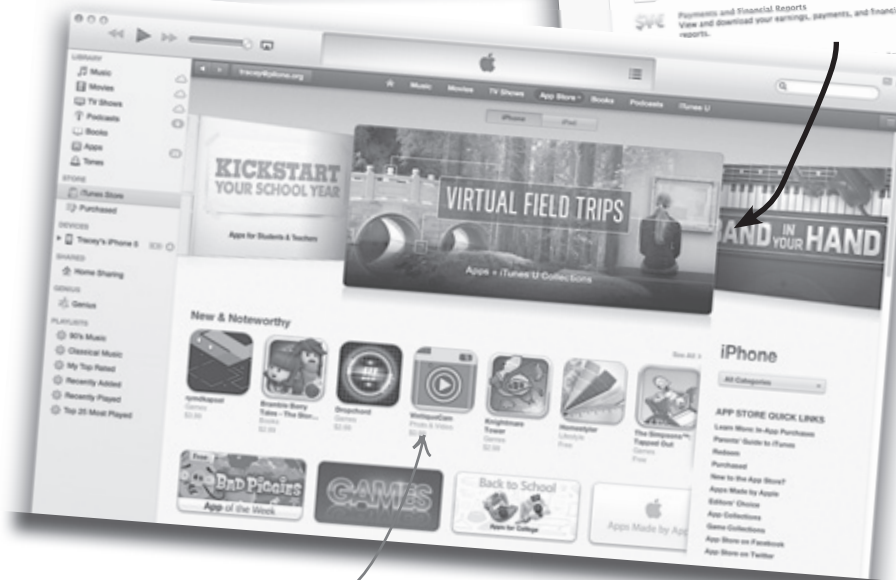
Добро пожаловать во Вселенную Apple!

Вероятно, вы уже в курсе, что Apple все делает по определенным правилам. iTunes, электронный магазин для приложений, был первым в своем роде. Чтобы ваше приложение продавалось в App Store, вы должны написать его в среде Xcode на Mac, а затем отправить через iTunes Connect (портал для отправки и отслеживания приложений iOS). Далее приложение должно быть одобрено экспертами Apple из Калифорнии, и только после этого оно появится в App Store.

Да, этим действительно занимаются живые люди. Мы общались с ними... чаще, чем хотелось бы.



Xcode



Ваше приложение

При разработке нужно учитывать множество мелочей, но все они складываются в общую картину. Давайте напишем простое приложение...

Приложения iOS пишутся на языке Objective-C

Вы уже работали на других языках программирования, так что многие концепции, с которыми нам придется иметь дело, покажутся знакомыми. Objective-C — объектно-ориентированный язык, созданный на базе C. Принципиально новых концепций не так много, и это хорошо. Впрочем, есть и плохие новости: синтаксис Objective-C на первый взгляд может показаться запутанным. Вероятно, вам придется какое-то время поломать голову. Не волнуйтесь, у вас все получится.



Objective-C похож на Java.

Объектно-ориентированный язык, уходящий корнями в Smalltalk.



Objective-C использует знакомый синтаксис.

Так как язык создавался на основе C, в нем используются знакомые по C конструкции циклов, типов, указателей и т. д. Objective-C продолжает линию наследия Apple, начавшуюся с системы NeXTStep, за которой последовала система OpenStep, и наконец, CocoaTouch.

В синтаксисе Objective-C часто встречается префикс NS. Знайте: это наследие NeXTStep!



Objective-C использует библиотеку CocoaTouch.

Если вы занимались программированием Mac, то вы уже знаете Objective-C. Впрочем, вам предстоит узнать еще немало того, что относится к специфике iOS.



Управление памятью может осуществляться автоматически.

В iOS5 компания Apple наконец-то представила механизм автоматизации управления памятью (по аналогии с Java), называемый автоматическим подсчетом ссылок (ARC, Automatic Reference Counting). А значит, вам уже не придется возиться с подсчетом ссылок для предотвращения утечки памяти.

Все начинается с SDK

Software Development Kit

Чтобы программировать на Objective-C, вам понадобится Mac и среда Xcode. Средства разработки распространяются бесплатно, а установить их несложно. Зайдите в магазин Mac App Store и найдите в нем Xcode. При написании книги использовалась версия Xcode 5.0.

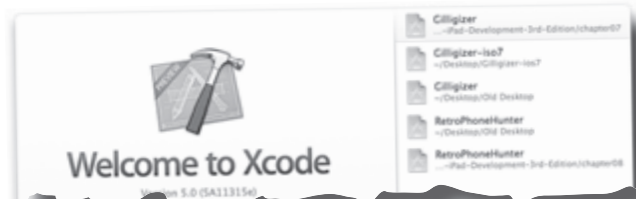


Найдите Xcode

Загрузите!

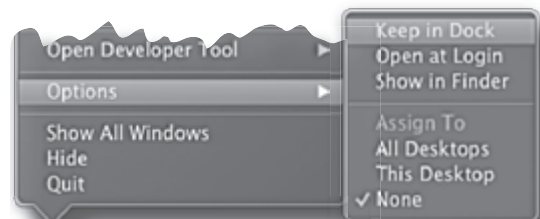


После загрузки Xcode помещается в папку Applications. В действительности Xcode представляет собой целый пакет, объединяющий множество приложений, включая компиляторы. О том, какие еще приложения входят в пакет Xcode, будет рассказано позднее, а пока просто запустите его. Для этого откройте папку Applications в Finder или воспользуйтесь Spotlight.



Закрепите Xcode на док-панели — пригодится на будущее

После того как вы впервые запустите Xcode, не поленитесь и добавьте значок Xcode на док-панель. Мы будем постоянно использовать среду разработки в книге. Щелкните на значке Xcode, удерживая клавишу Ctrl, выберите команду «Options», а затем команду «Keep in Dock».



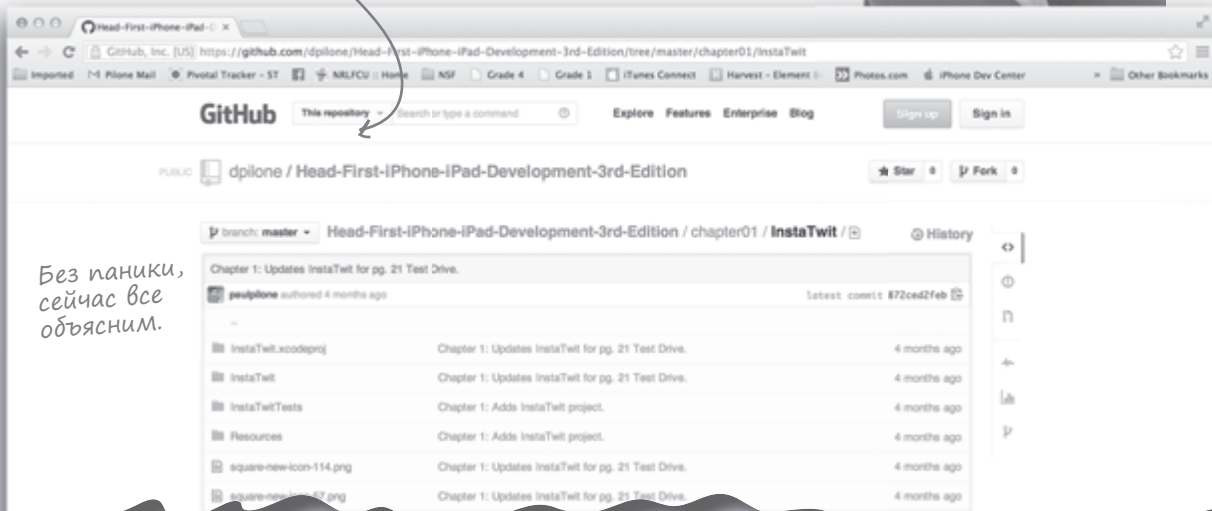
Это Сью, ваш новый начальник

Вы только что поступили на работу в новую фирму, занимающуюся разработкой для iOS, и получили свое первое задание: внести изменения в почти готовое приложение для Twitter. Мы непременно расскажем все, что для этого необходимо знать, а вам представился хороший шанс отличиться.

Весь наш код хранится в *GitHub*, так что загружай *InstaTwit* и берись за дело. Я расскажу, что нужно исправить, чтобы мы могли выпустить приложение. Отдел маркетинга хочет внести пару изменений.

А это *GitHub*

Сью из тех начальников, которые считают, что новичок должен сразу браться за дело...



Без паники, сейчас все объясним.

Часть Задаваемые Вопросы

В: Что такое *GitHub*?

О: *GitHub* — площадка для «социального программирования». Это сайт, на котором размещаются миллионы репозиторий *Git*, многие из которых являются общедоступными. *GitHub* отлично подходит для распределенных команд и проектов с открытым кодом. И еще

здесь удобно хранить код для книг. Вы найдете на *GitHub* все примеры кода из книги, включая код этого проекта. Для управления исходным кодом используется система *Git*; это означает, что вы можете загрузить копию кода, необходимого для работы с книгой, и отслеживать все вносимые изменения. А если что-то пойдет не так, то неудачное изменение всегда можно будет отменить и вернуться к предыдущей версии.

Xcode и Git... неразлучные друзья

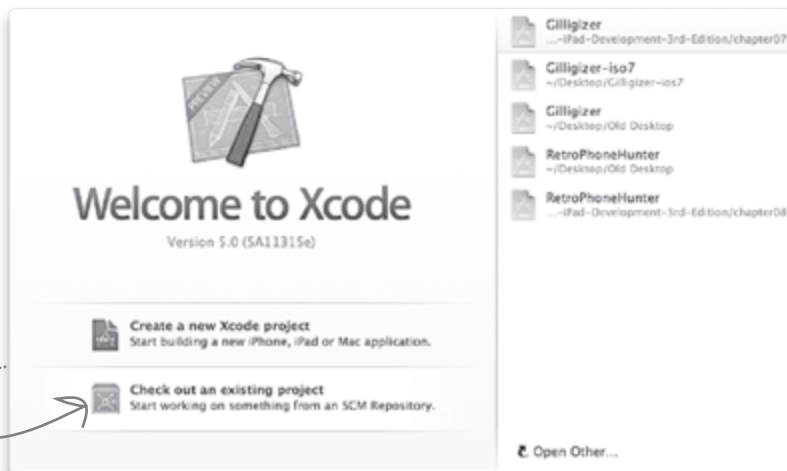
Xcode — среда разработки для создания приложений iOS. GitHub — сетевой репозиторий кода программных проектов (использующий Git). Эти инструменты упрощают жизнь групп разработки: вы работаете над проектом самостоятельно, но используете код, написанный другими участниками. Учетная запись GitHub для этого не потребуется, но если она у вас есть — не стесняйтесь, создавайте новую ветвь и используйте код.

1

Не создавайте новый проект — он уже существует.

На начальном экране Xcode вам предлагается создать новый проект или подключиться к уже существующему. В данном случае следует выбрать второй вариант («Check out an existing project»).

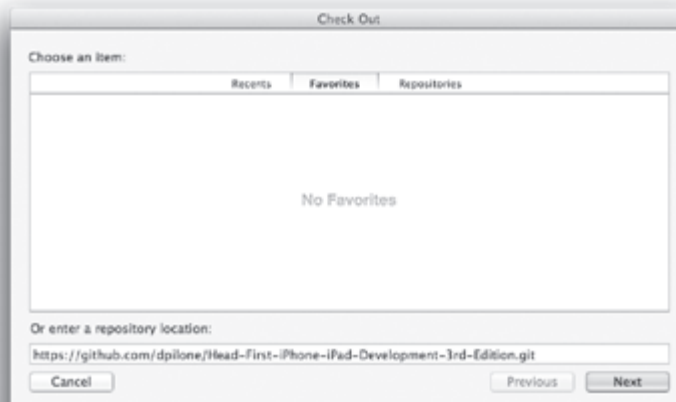
Выберите
этот вариант.



2

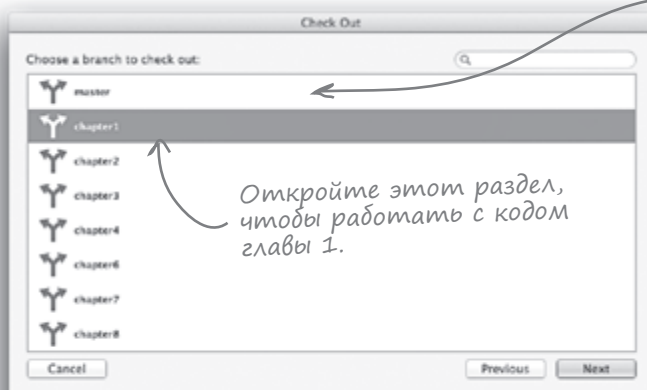
Клонируйте содержимое репозитория.

Далее необходимо сообщить Xcode местонахождение кода. В нашем случае введите адрес <https://github.com/dpilone/Head-First-iPhone-iPad-Development-3rd-Edition.git>. Введите адрес в нижнем поле и щелкните на кнопке Next.



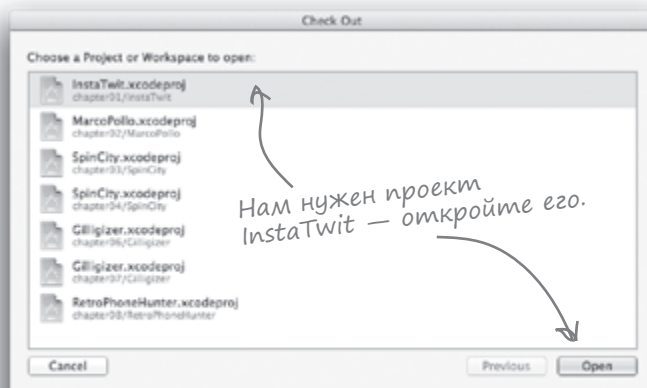
3 Выберите нужный вариант.

Код книги организован так, что вы можете либо загрузить код отдельной главы, либо полный архив кода всей книги. Чтобы открыть код конкретной главы, откройте соответствующий раздел (в данном случае главу 1).



4 Выберите проект.

Раздел содержит несколько проектов, но в данный момент нас интересует только проект InstaTwit. Выберите его и щелкните на кнопке Open.



Сетевой репозиторий **GitHub** используется многими проектами — как закрытыми, так и распространяемыми с открытым кодом.

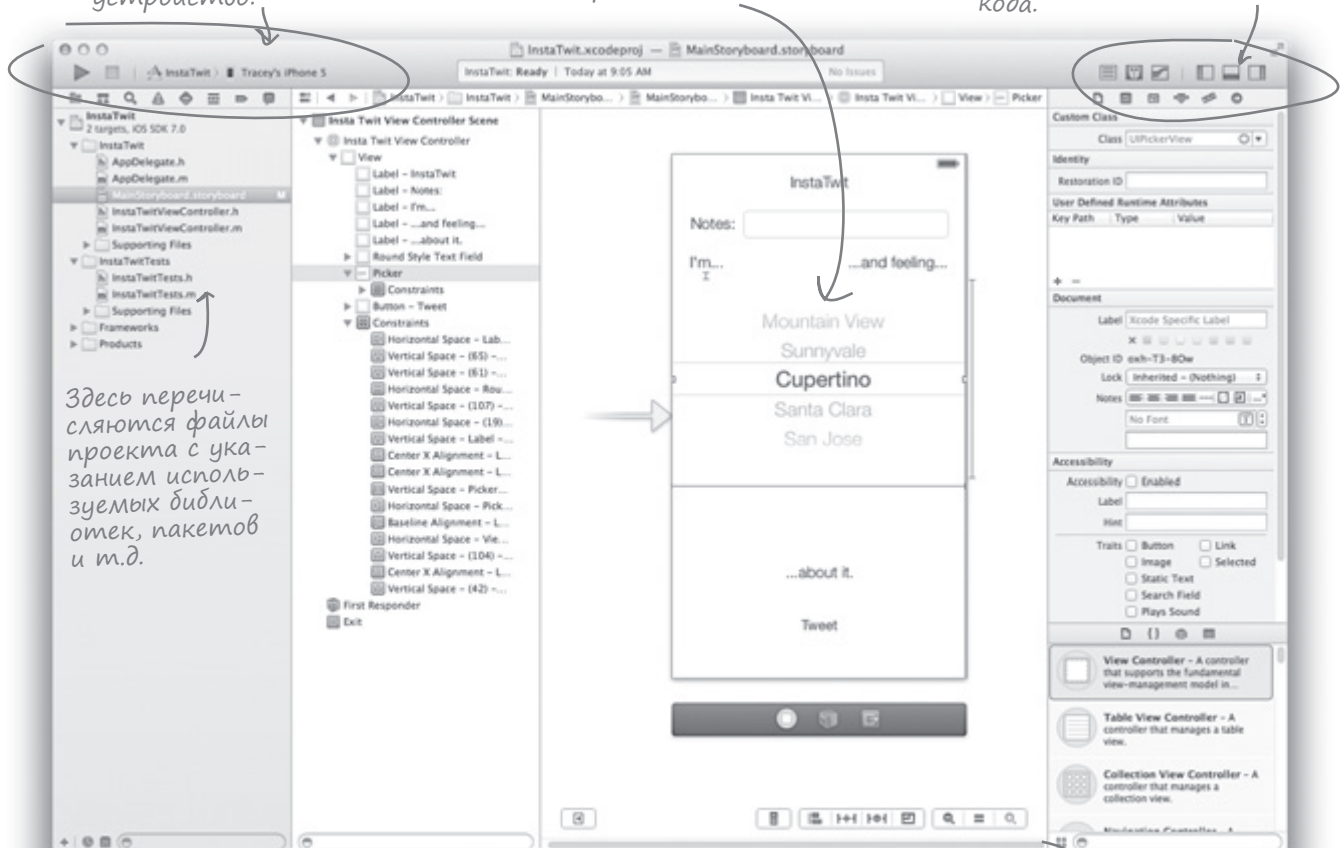
Xcode находится в центре любого проекта iOS

Xcode — полнофункциональная интегрированная среда разработки (IDE). Разработчик использует Xcode для редактирования кода и построения пользовательских интерфейсов, а также для работы с документацией. В Xcode входит полноценный отладчик, компилятор, поддержка контекстного редактирования, статический анализатор кода, система выдачи рекомендаций и предупреждений по качеству кода, а также средства контроля версий на базе Git или Subversion.

Здесь выбирается модель построения; пока мы будем использовать эмулятор, потом перейдем на физическое устройство.

Здесь в Xcode отображается редактор графического интерфейса.

Элементы управления для настройки представления, включая добавление консоли и вспомогательных редакторов для вывода кода.



Здесь перечисляются файлы проекта с указанием используемых библиотек, пакетов и т.д.



На вашем компьютере Xcode пока выглядит не так.

На иллюстрации изображена среда Xcode в самом разгаре редактирования, с несколькими включенными представлениями. Пока не обращайте внимания на различия.

Библиотека объектов Xcode: отсюда элементы перетаскиваются мышью в представление.

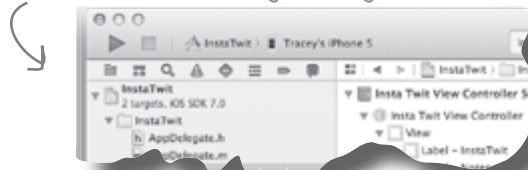


ТЕСТ-ДРАЙВ

1 Постройте и запустите код проекта.

Убедитесь в том, что для проекта выбрана схема эмулятора «InstaTwit» iPhone Retina (4-inch).

Чтобы программа заработала, достаточно нажать одну кнопку!



2 Щелкните на кнопке Build and Run.

Она выглядит как кнопка воспроизведения на бытовых проигрывателях.

3 Включите режим разработчика.

В этот момент на экране может появиться окно с предложением включить режим разработчика. Включите его — иначе вам придется снова и снова вводить свой пароль!

Часто задаваемые вопросы

В: Еще раз — что такое Git?

О: Git — система управления версиями, изначально созданная для разработки Linux. Она отличается от Subversion или CVS тем, что каждая локальная копия репозитория сам по себе является полным репозиторием. По этой причине система Git более надежна в условиях распределенных групп асинхронной разработки. Дополнительная информация о системе Git и ее командах приведена по адресу <http://git-scm.com/about>.

В: И как Git относится к GitHub?

О: Git — система управления версиями, которая может устанавливаться локально на одном компьютере или на приватном сервере для удаленного доступа. GitHub — общедоступный сетевой репозиторий для проектов Git.

В: А как же Subversion?

О: Subversion также поддерживается в Xcode. Если вы работаете с Subversion,

вы можете использовать тот же рабочий процесс (и URL-адрес), но в книге система Subversion не задействована, и для кода InstaTwit необходимо использовать Git.

В: Как еще среда Xcode связана с Git?

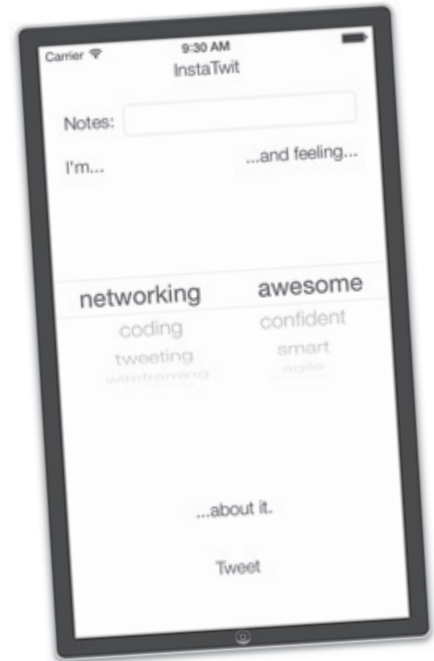
О: Xcode использует Git для отслеживания изменений в файлах. Измененные файлы помечаются специальным значком; разработчик может закрепить изменения прямо из Xcode и даже сравнить старые файлы с новыми.

Эмулятор iOS

Эмулятор запускается автоматически при первом построении и запуске приложения. Он поддерживает основные операции с устройством, не ограничивающиеся простыми прикосновениями к экрану, включая повороты, создание снимков экрана для отправки приложения на рецензирование и встряхивание устройства для проверки акселерометра.

Представление, отображаемое в эмуляторе iOS, — это отдельное приложение, которое запускается автоматически.

И хотя изображение напоминает экран телефона, эмулятор iOS также поддерживает iPhone 4s, iPhone 5s, а также iPad с экраном Retina (и без него).



Я-то думала, что будет сложно! Но стоило нажать пару кнопок, и все заработало!

Среда Xcode выполнила за вас немалую работу.

Хотя приложение запускается всего одной кнопкой, в Xcode при нажатии этой кнопки происходит много всего. Хотя мы не написали ни строчки нового кода, у вас уже был готовый проект. Когда вы нажали кнопку, среда Xcode откомпилировала, скомпоновала, обработала, упаковала, установила и запустила приложение. Давайте повнимательнее разберемся, что же произошло...



Построение в Xcode: под увеличительным стеклом

Формально это не является частью процесса построения, но мы не хотели обходить вниманием вопрос о том, откуда взялся код...

1

Загрузка проекта с GitHub.

Выше этот шаг подробно не описывался, но среда Xcode загрузила исходный код с GitHub. Впрочем, загружается не только код на языке Objective-C, но также полное описание проекта, макет рабочей области, значки и графические элементы, используемые в приложении. Все эти данные являются частью проекта Xcode.

2

Компиляция классов проекта.

Класс Objective-C обычно состоит из файлов *.h* и *.m*, содержащих заголовки и реализацию соответственно. Файл заголовка описывает открытый интерфейс класса — его API. Файл реализации содержит «суть» класса: то, благодаря чему класс выполняет свою работу. Эти два файла объединяются и компилируются в двоичную форму в процессе построения.



3

Обработка пользовательского интерфейса и его преобразование в двоичную форму.

Пользовательский интерфейс в Xcode обычно строится в графическом редакторе, и для него создается документ XML с описанием макета. Xcode компилирует описание в двоичный формат и оптимизирует графику, значки и т. д., используемые приложением, в формат, хорошо подходящий для отображения на целевом устройстве.

4

Компоновка кода библиотек.

В приложениях iOS используется большой объем кода общих библиотек: отображения пользовательского интерфейса (UIKit), картографической поддержки (MapKit), низкоуровневого кода графического вывода (CoreGraphics) и т. д. Apple называет эти общие библиотеки инфраструктурами (frameworks). Инфраструктуры содержат не только общий откомпилированный код (как, например, файлы JAR или *.so* в других языках). В Objective-C инфраструктуры могут содержать заголовочные файлы, графические ресурсы и т. д. Некоторые инфраструктуры — такие, как MapKit, — подключаются только в том случае, если они используются приложением; другие — такие, как CoreGraphics, — являются обязательными, поскольку все приложения зависят от содержащегося в них кода.



Построение в Xcode: под увеличительным стеклом (продолжение)

5

Упаковка откомпилированного кода, графики и т. д.

Если вы никогда не работали с приложениями OS X или iOS, знайте, что *SomeApp.app* — всего лишь каталог с особым содержимым. Приложение iOS представляет собой папку с откомпилированным исполняемым файлом, набором конфигурационных файлов и всеми ресурсами, необходимыми для работы приложения. Структура каталогов более подробно рассматривается ниже.

6

Если приложение устанавливается на устройстве, код снабжается цифровой подписью.

Установка на устройстве — во всех случаях, от простого тестирования на вашем личном устройстве до отправки в App Store — проходит под управлением сертификатов, сгенерированных на базе ваших удостоверений разработчика. Так экосистема iOS следит за тем, какие программы могут устанавливаться на устройствах. Для управления сертификатами используется служебная программа-организатор, а для сопоставления приложений с устройствами — система профилей.



Все профили в установке.

Пока приложение запускается в эмуляторе, так что этот шаг для нас еще не актуален.

7

Приложение устанавливается в эмуляторе.

Эмулятор ведет себя как реальное устройство. Он тоже содержит каталог для хранения данных, приложения тоже создают свои каталоги с использованием идентификаторов, и т. д. Мы еще поговорим об этом позднее, но помните: все это происходит при нажатии кнопки Build and Run...

8

Приложение запускается, и к нему присоединяется отладчик.

Xcode запускает приложение в эмуляторе и присоединяет к нему отладчик, если вы решили запустить приложение в отладочном режиме (вместо простого запуска без отладки). События, происходящие в приложении, начинают выводиться на консоль, и вы можете просмотреть их на панели в Xcode.

И правда, немало...

далее ▶

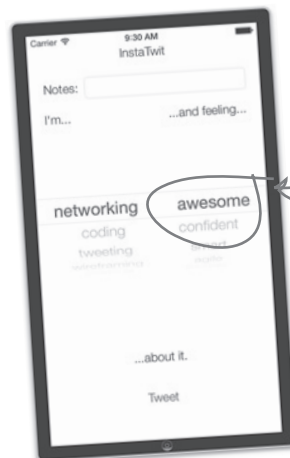




Хорошо — я вижу, у тебя все готово. Нам нужно изменить текст сообщений — отдел маркетинга хочет представить наше приложение как средство экономии времени.

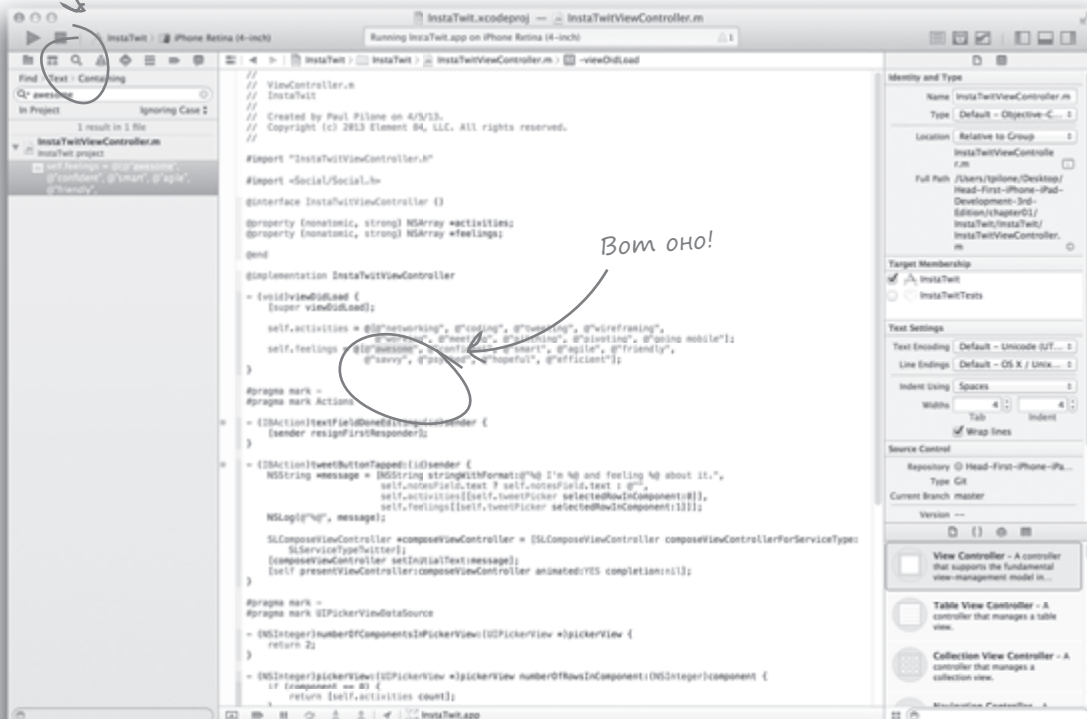
Поиск помогает в решении многих проблем.

Это изменение не столь значительное (мы изменяем статический текст, жестко запрограммированный в приложении). Как проще всего определить, где находится код с нужным текстом? Конечно, воспользоваться поиском. Мы будем искать слово «awesome», встречающееся в одной из надписей.



Условие поиска.

Щелкните здесь, чтобы открыть диалоговое окно поиска.



Вот оно!

Программа хранится в файлах с исходным кодом

Код приложения делится на классы, а каждый класс состоит из двух файлов: заголовочного файла (*.h*) и файла реализации (*.m*). Заголовочный файл (*.h*) определяет открытый программный интерфейс (API) класса, тогда как файл реализации (*.m*) определяет, как именно класс делает то, что полагается. Наша задача относится к данным, выводимым классом, а не к его API.

Один из методов реализации — метод `viewDidLoad`.

```
(void)viewDidLoad {
    [super viewDidLoad];

    self.activities_ = @[@"sleeping", @"eating", @"working",
                        @"thinking", @"crying", @"begging", @"leaving", @"shopping",
                        @"hello worlding", nil];

    self.feelings_ = @[@"awesome", @"sad", @"happy",
                     @"ambivalent", @"nauseous", @"psyched", @"confused", @"hopeful",
                     @"anxious", nil];
}
```

Этот метод вызывается инфраструктурой непосредственно после загрузки представления. Он предназначен для настройки представления перед его отображением. В данном случае два массива заполняются значениями, которые отображаются на спиннере.

В этом простом приложении текст просто сохраняется в обычном массиве, но есть и другие варианты.

InstaTwitViewController.m

О синтаксисе мы еще поговорим позднее, а пока достаточно сказать, что перед вами массив строк Objective-C — экземпляров класса `NSString`.



InstaTwitViewController.h — заголовочный файл этого класса; он объявляет открытый интерфейс класса, то есть его общедоступные свойства и действия. Если вам интересно, откройте заголовочный файл и посмотрите сами.

Упражнение!



Внесите следующие изменения в кодовую базу.

Код, который нужно удалить, в книге будет обозначаться перечеркиванием.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.activities = @[@"sleeping", @"eating", @"working", @"thinking",
@"crying", @"begging", @"leaving", @"shopping", @"hello worlding",
@"networking", @"coding", @"tweeting", @"wireframing", @"working",
@"meeting", @"pitching", @"pivoting", @"going mobile", nil];
    self.feelings = @[@"awesome", @"sad", @"happy", @"ambivalent",
@"nauseous", @"psyched", @"confused", @"hopeful", @"anxious", @"awesome",
@"confident", @"smart", @"agile", @"friendly", @"savvy", @"psyched",
@"hopeful", @"efficient", nil];
}
```

Полужирным шрифтом обозначается добавляемый код.

Записка от Сью с новыми терминами

Текст, который нужно изменить:

- Старый текст удаляем, заполняем массивы новыми данными...
- Mассив activities — networking, coding, tweeting, wireframing, working, meeting, pitching, pivoting, going mobile
- Mассив feelings — awesome, confident, smart, agile, friendly, savvy, psyched, hopeful, efficient



InstaTwitterViewController.m

Часть Задаваемые Вопросы

В: Мне всегда придется строить приложение, чтобы его протестировать?

О: Да, некоторые языки (например, Ruby и JavaScript) обходятся без компиляции, но Objective-C — компилируемый язык. Чтобы протестировать изменения, необходимо построить приложение, установить его на эмулятор (или реальное устройство) и запустить. В Xcode реализована качественная поддержка модульного и функционального тестирования, а служебная программа Instruments может использоваться для полного тестирования взаимодействий с приложением. Автоматизированные тесты ускоряют цикл тестирования и избавляют от необходимости построения и запуска всего приложения, но не заменяют комплексного тестирования.

В: Насколько важно протестировать программу на реальном устройстве? Можно ли ограничиться эмулятором?

О: Эмулятор способен на многое. Большую часть тестов вы будете выполнять в эмуляторе (впрочем, это зависит от приложения). И все же существуют различия между запуском программы в эмуляторе и на реальном устройстве. В какой-то момент придется перейти на устройство, причем лучше сделать это рано, чем поздно. О том, какие возможности не поддерживаются эмулятором, будет рассказано в следующих главах. Но даже если вы уверены, что эмулятор поддерживает нужные функции, тестирование производительности **всегда** следует проводить на реальном устройстве — никогда не полагайтесь на производительность эмулятора при тестировании приложений.

Возьми в руку карандаш



Что делают эти методы? Вы уже знаете, что делает метод `viewDidLoad`; теперь взгляните на другие методы из файла `.m` и напишите, что они, по вашему мнению, делают.

```
- (IBAction) textFieldDoneEditing: (id) .....
sender { .....
    [sender resignFirstResponder]; .....
} .....
```

```
- (NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent:(NSInteger)component {
    if (component == 0) {
        return [activities_ count]; .....
    } .....
    else {
        return [feelings_ count]; .....
    }
}
```

```
- (IBAction) tweetButtonTapped: (id) sender {
    NSLog(@"%@", @"Tweet button tapped!");
    NSString...
```

Это не весь код метода.

Это не весь код метода. Чтобы просмотреть его продолжение, воспользуйтесь Xcode.



РАССЛАБЬТЕСЬ

Если код Objective-C выглядит непривычно, не огорчайтесь...

Мы всего лишь хотели дать общее представление о структуре кода Objective-C. Если вы в общих чертах понимаете, что здесь происходит, — хорошо, но в книге Objective-C будет рассматриваться более подробно. Бояться пока нечего.

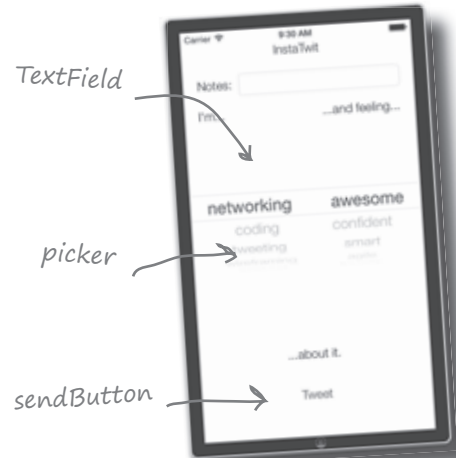
Возьми в руку карандаш

Решение

Некоторые методы из файла *InstaTwitViewController.m*.

```
- (IBAction) textFieldDoneEditing: (id)
sender {
    [sender resignFirstResponder];
}
```

Это метод обратного вызова, который присоединяется к текстовому полю (вскоре вы узнаете, как это делается...). Когда пользователь завершает редактирование текстового поля, компонент отправляет оповещение *resignFirstResponder*. Позднее статус «первого ответчика» (*FirstResponder*) будет рассмотрен более подробно.



```
- (NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent: (NSInteger) component {
    if (component == 0) {
        return [activities_ count];
    }
    else {
        return [feelings_ count];
    }
}
```

Компонент выбора запрашивает у контроллера представления, сколько строк в нем должно отображаться. В нашем случае оно зависит от количества элементов в массивах *activities* и *feelings*.

```
- (IBAction) tweetButtonTapped: (id) sender {
    NSLog(@"%@@", @"Tweet it button
tapped!");
    NSString...
```

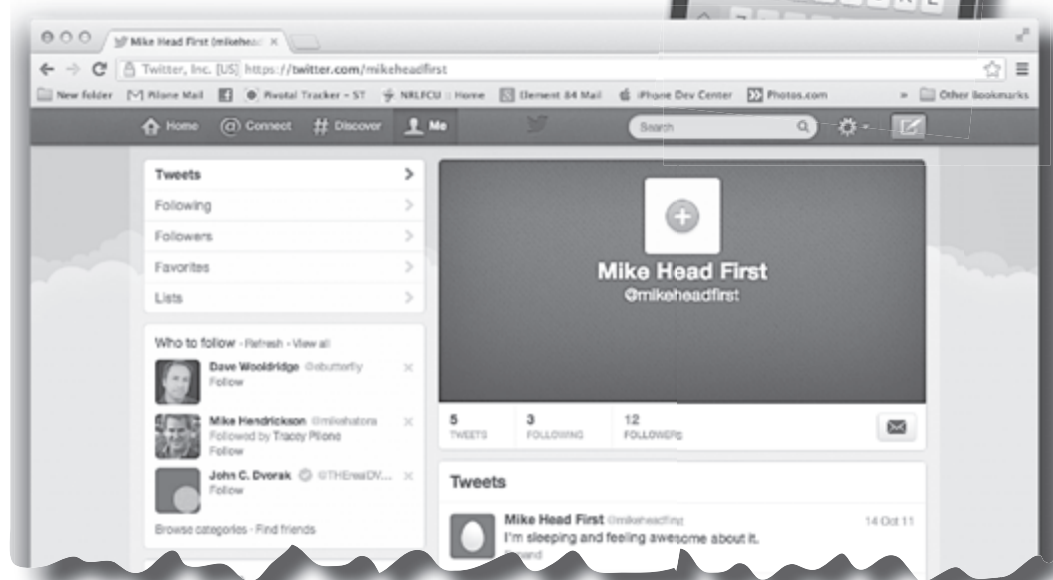
При нажатии кнопка должна что-то делать. Как и в случае с текстовым полем, кнопка иницирует действие, с которым связывается фрагмент кода. В нашем примере код выводит в журнал комментарий. О том, как просмотреть содержимое этого журнала, будет рассказано далее. (Код публикации в Twitter в листинге не показан.)



ТЕСТ-ДРАЙВ

Итак, мы нашли код, который необходимо изменить в данном приложении. Внесите изменения и постройте приложение заново.

- 1 **Запустите Xcode и внесите изменения в текст.**
Вернитесь на пару страниц назад к файлу *InstaTwitViewController.m* и внесите изменения, показанные на странице.
- 2 **Сохраните приложение и постройте его заново.**
После изменения текста повторите процедуру построения и запуска.
- 3 **Настройте учетную запись Twitter в эмуляторе.**
Если вы хотите, чтобы сообщения действительно появлялись в Twitter, нажмите кнопку None в эмуляторе, перейдите к настройкам и введите данные своей учетной записи. Затем вернитесь к приложению и отправьте сообщение!



Xcode может существенно помочь с отладкой, если в приложении возникли проблемы....

Редактор кода, центр управления... да еще и отладчик

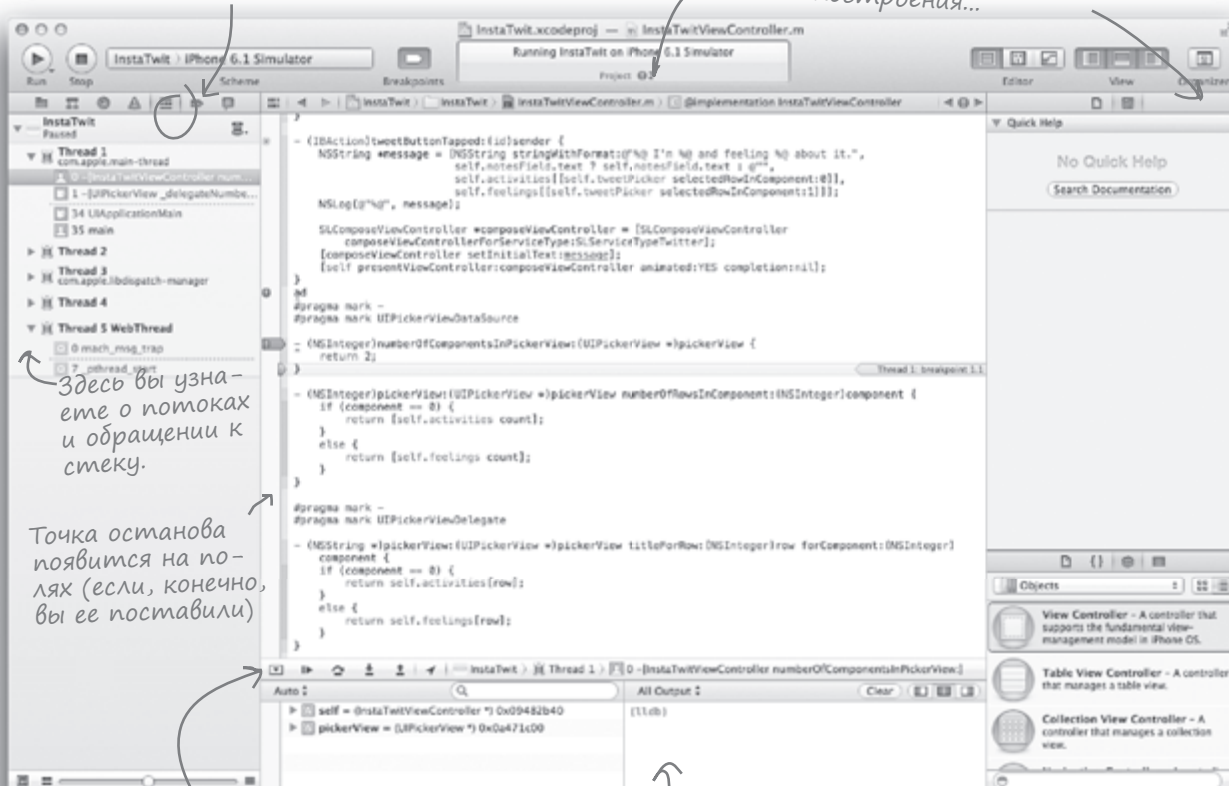
Каждый из нас хоть иногда да ошибается при вводе или не замечает, когда его код остается незавершенным. Наверняка и вы сталкивались с ошибками и предупреждениями в редакторе. Xcode старается следить за происходящим во время написания кода и выдает ошибки и предупреждения еще до того, как вы перейдете к построению программы.



Переключите навигатор в потоковое представление.

Здесь выводятся предупреждения и ошибки построения...

...и здесь тоже



Элементы управления для запуска приложения в отладочном режиме: запуск и приостановка, пошаговое выполнение с обходом процедур, пошаговое выполнение с заходом в процедуры и т. д.

Здесь выводится дополнительная информация о том, что именно происходит в приложении.

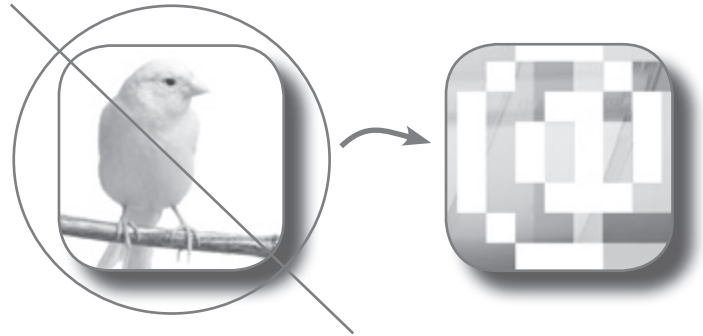
Когда программа заработает, вы также можете воспользоваться полнофункциональным отладчиком Xcode для выявления возможных проблем. Вы можете устанавливать точки прерывания, щелкая на боковых полях, включать отслеживание переменных, а при достижении программой одной из установленных точек прерывания — проверять значения переменных и выполнять код в пошаговом режиме.



Отлично! Осталась одна мелочь. Отдел маркетинга прислал новый значок приложения. Заменим его — и можно подавать на утверждение.

Обратили внимание на старый значок?

Никакой фантазии у людей, все та же унылая птичка. Чтобы привлечь внимание технарей, отдел маркетинга предлагает использовать символ @ в модном восьмибитном ретро-стиле.



Xcode позволяет легко сменить значок приложения.

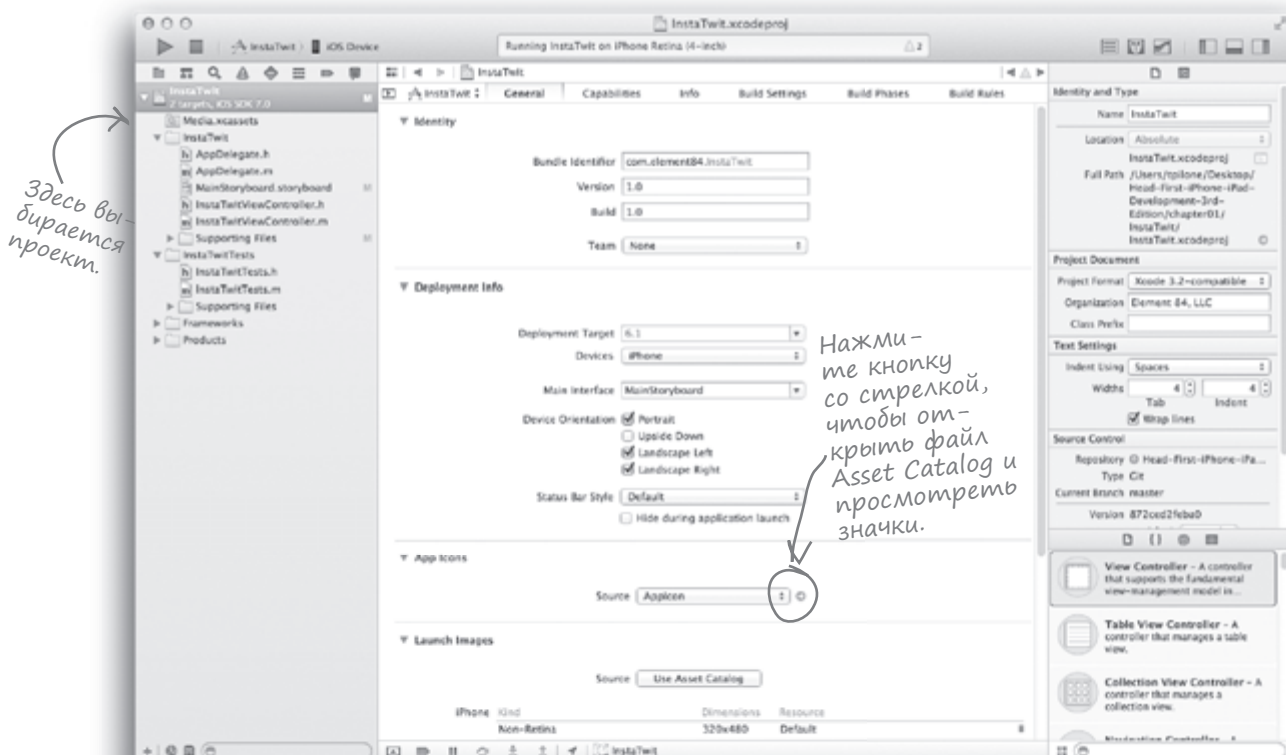
В Xcode графика рассматривается как ресурсы приложения. С каждым приложением (даже с этим) связываются различные ресурсы, от значков до изображений высокого разрешения и встроенных видеороликов.

Xcode помогает упорядочить изображения (с возможностью просмотра многих видов графики), а затем упаковывает и осуществляет предварительную обработку для использования в приложении.

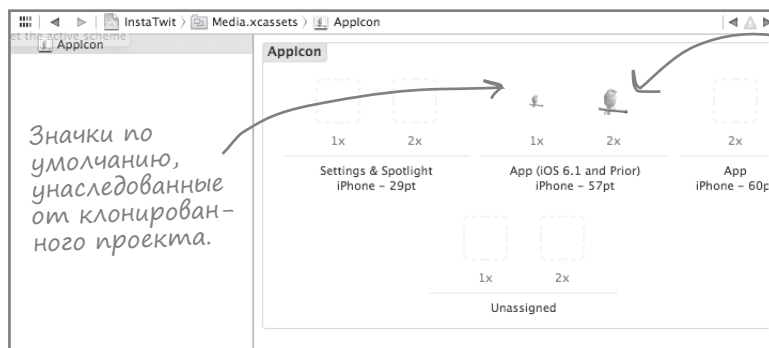
Один iPhone, два iPhone, красный iPhone, синий iPhone...

В Xcode щелкните на имени главного проекта, чтобы вызвать страницу со сводной информацией. На этой странице отображается различная информация высокого уровня: данные развертывания, некоторые изображения, номера версий. Кроме того, с этой страницы можно перейти к каталогу ресурсов (Asset Catalog) приложения.

Ладно, мы слегка преувеличили, но друг моего друга говорил, что знает одну женщину, которая говорила, что Apple работает над светящимися в темноте моделями...



Каталоги ресурсов — это специализированные файлы для хранения графики непосредственно в приложении. На этой странице отображается удобный графический интерфейс для управления значками. Аналогичные интерфейсы используются для управления заставками и прочей графикой.



Конечно, различия между iPhone и iPad не ограничиваются размером значков? Давайте угадаю — скоро вы об этом расскажете подробнее?

Безусловно! Мы будем разбираться с различиями устройств по ходу дела.

Разные устройства и разные поколения одного устройства обладают разными возможностями. iOS обычно старается скрыть такие подробности от пользователя. Например, при построении макета пользовательского интерфейса на экране Retina iOS будет автоматически выводить графику в более высоком разрешении. Тем не менее некоторые моменты желательно принимать во внимание: различия в размерах значков, наличие или отсутствие камеры и т. д. Мы будем попутно упоминать о них.



ТЕСТ-ДРАЙВ

- 1 **Найдите значки из клонированного проекта.**
Код, клонированный из GitHub, содержит дополнительные значки. Воспользуйтесь Finder и перейдите к месту сохранения клонированного кода.
- 2 **Перетащите файлы в Asset catalog.**
Возьмите новые значки и перетащите их из Finder. Файлы значков называются *square-new-icon-60.png* и *square-new-icon-120.png*.
- 3 **Удалите приложение из эмулятора.**
Чтобы убрать старый значок, войдите в эмулятор, нажмите кнопку Home и щелкните на значке InstaTwit. Когда значок начнет дрожать и появится маленькая кнопка «х», щелкните на ней для удаления приложения.
- 4 **Постройте и запустите приложение в Xcode.**
Снова щелкните на кнопке запуска. Приложение строится и заново устанавливается в эмуляторе с новым значком. Нажмите кнопку Home, и вы увидите его!





ТЕСТ-ДРАЙВ

Новый эффектный значок появляется на домашнем экране!



КЛЮЧЕВЫЕ МОМЕНТЫ



- Xcode занимает центральное место в разработке для iOS.
- Xcode — мощная интегрированная среда с документацией, средствами отладки и редакторами пользовательского интерфейса.
- Эмулятор удобен, но у него есть свои ограничения; тестирование следует проводить на реальном устройстве.
- Для установки приложения на устройство необходимо быть участником платной программы Apple Developer.
- Поддержка Git встроена в Xcode.
- Помните о различиях в функциональности устройств при построении приложения.

А как было бы замечательно, если бы приложение можно было установить на iPhone вместо эмулятора? Но это, конечно, только мечты...





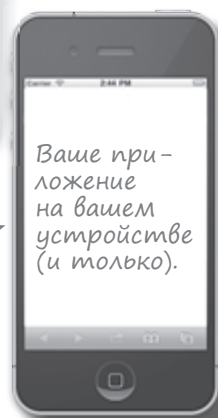
Это возможно. Только заплатите \$99 компании Apple.

И это не шутка. Чтобы установить приложение на устройство, необходимо решить ряд задач, связанных с безопасностью, создать необходимые сертификаты и профили. Для доступа к этим механизмам необходимо зарегистрироваться в платной программе разработчиков для iOS. За дополнительной информацией обращайтесь по адресу: <http://www.apple.com/iosdeveloper>.

В этой книге мы ограничимся эмулятором, так что читателям платить за участие в программе не обязательно.



Вы находитесь
здесь.



Ваше при-
ложение
на вашем
устройстве
(и только).

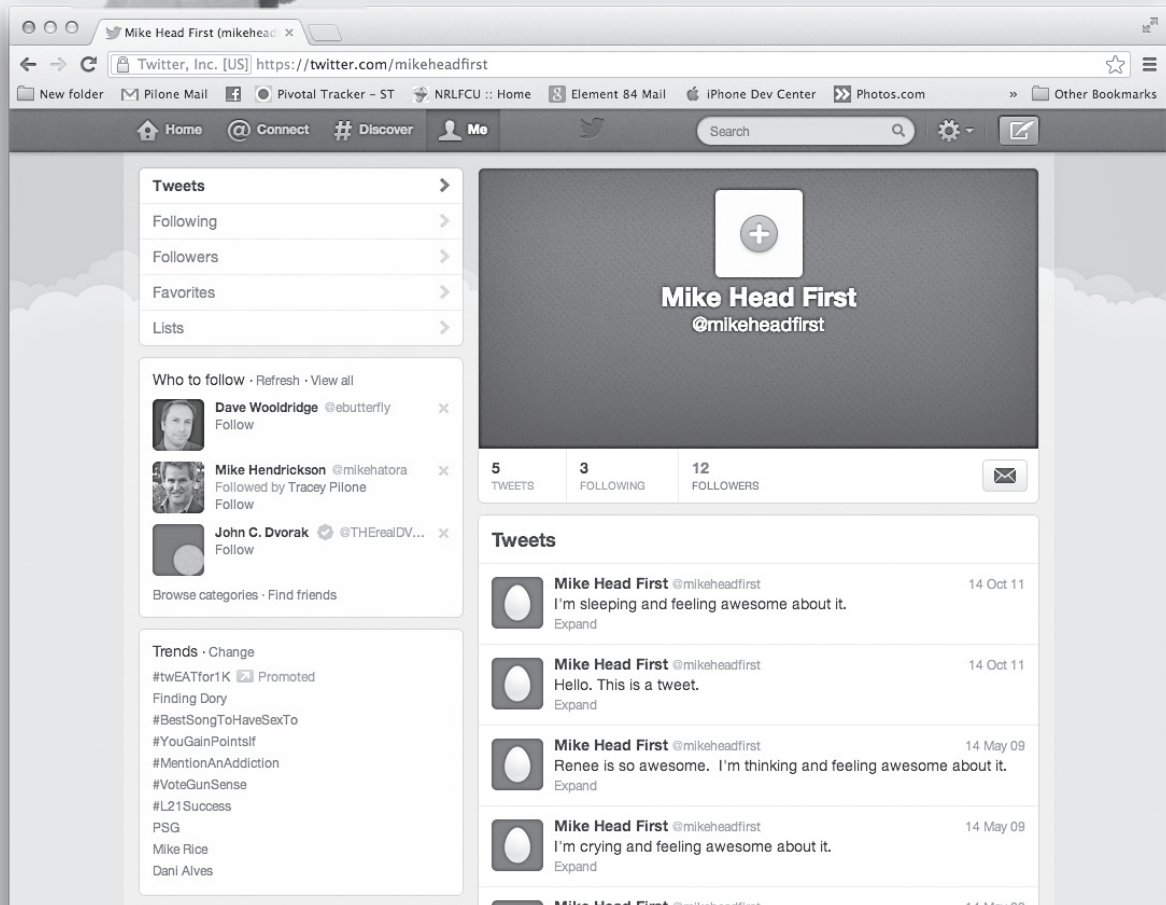
Хотите большего? Заходите на сайт!

Я просмотрела изменения —
все отлично. Отправляйте!



Прекрасная работа!

Вы попробовали себя в деле с Xcode и начали понемногу осваиваться в мире разработки Apple. Используя стандартные элементы управления и Xcode, можно быстро создавать вполне достойные приложения. Вы не представляете, как приятно видеть людей, работающих с вашим приложением... а впрочем, все только начинается.





Ваш инструментарий разработки для iPhone

Глава 1 осталась позади, а ваш инструментарий пополнился основными концепциями программирования для iOS.

Xcode

- ☐ Полнофункциональная интегрированная среда разработки (IDE) для программирования iOS и Mac.
- ☐ Управляет вашим кодом, подключаемыми инфраструктурами и ресурсами приложения.
- ☐ Встроенный редактор пользовательского интерфейса для построения макета представления и связывания компонентов.

Приложения iOS

- ☐ Должны подчиняться правилам и ограничениям Apple.
- ☐ Проходят процесс рецензирования перед допуском в App Store.
- ☐ Должны иметь цифровую подпись для установки на физическом устройстве.
- ☐ Пишутся на языке Objective-C.

Эмулятор

- ☐ Прекрасно подходит для быстрого тестирования.
- ☐ Обладает ограниченной поддержкой тестирования акселерометра, памяти, производительности, GPS и камеры.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Xcode занимает центральное место в разработке для iOS.
- Xcode — мощная интегрированная среда с документацией, средствами отладки и редакторами пользовательского интерфейса.
- Эмулятор удобен, но у него есть свои ограничения; тестирование следует проводить на реальном устройстве.
- Для установки приложения на устройство необходимо быть участником платной программы Apple Developer.
- Поддержка Git встроена в Xcode.
- Помните о различиях в функциональности устройств при построении приложения.

2 Основные паттерны iOS

Строим приложение «с нуля»

Да не так уж много времени
понадобилось. Пара дней на
подготовку — и дело в шляпе...

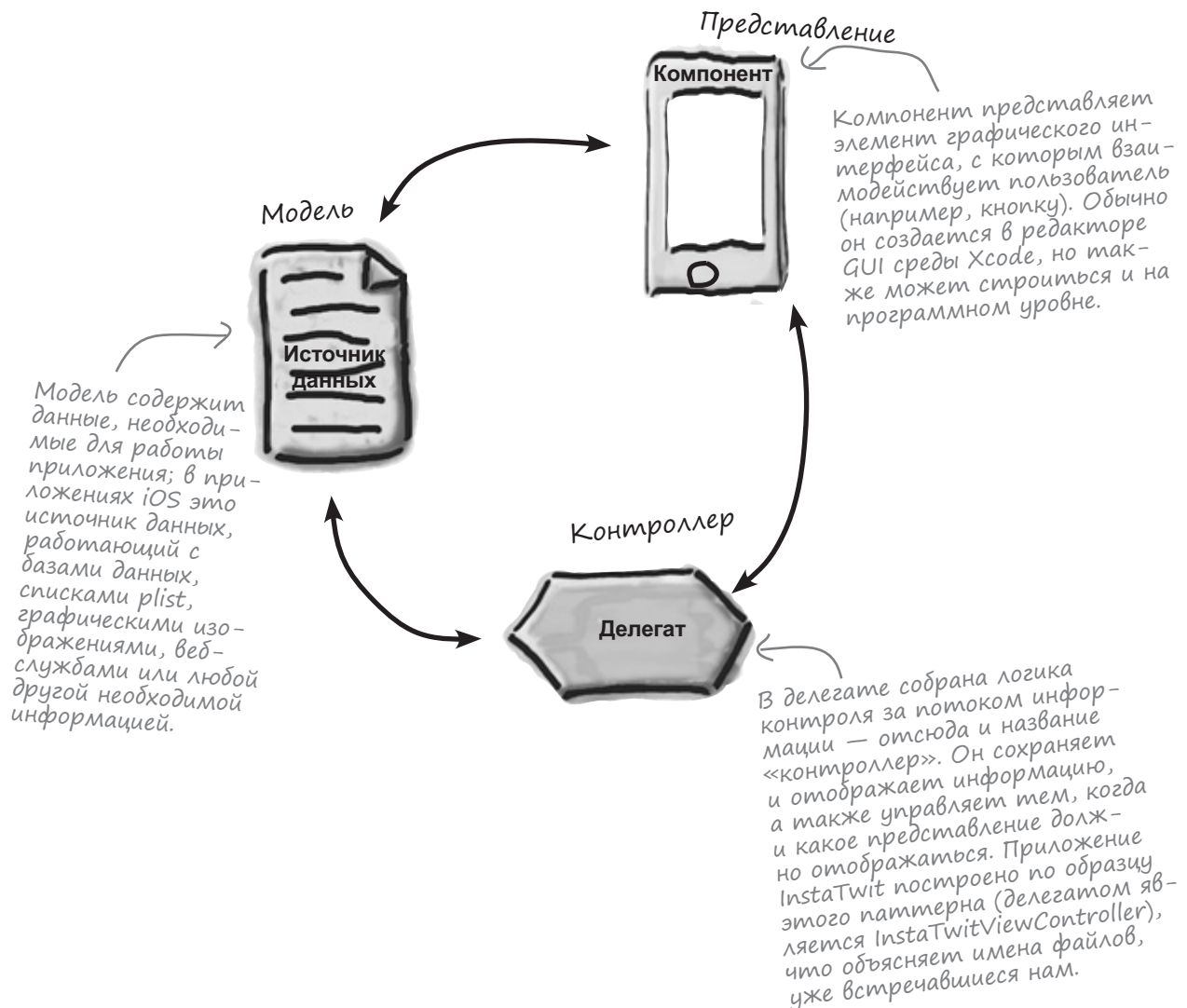


Итак, первое знакомство состоялось; теперь пришло время начать все заново. Вероятно, вы уже примерно представляете, с какими инструментами вам предстоит работать и как устроена среда Xcode. Пора изучить проблему более основательно и построить собственный проект. Как создать проект iOS, как связаны друг с другом компоненты приложений, на какие схемы взаимодействий следует ориентироваться? Вы это скоро узнаете — а пока переверните страницу...

Приложения iOS работают в полноэкранном режиме

Одно из принципиальных отличий устройств iOS (например, iPhone или iPad) перед настольными и портативными компьютерами заключается в том, что их приложения всегда работают в полноэкранном режиме. Более того, в первых моделях iPhone никакой многозадачности не было вообще. Впрочем, со временем она появилась, но это не та многозадачность, которая знакома нам по настольным компьютерам. Пользователь в любой момент времени взаимодействует только с одним приложением через представление, которое отображается на экране.

При этом приложение незаметно для пользователя распределяет обязанности по работе приложения между парой функциональных блоков, изображенных на следующей диаграмме.



Паттерн Модель-Представление-Контроллер

Паттерн «Модель-Представление-Контроллер» (MVC, Model View Controller) подробно рассматривался в книге Э. Фримена «Паттерны проектирования» (СПб., Питер, 2011). Он часто встречается в приложениях с графическим интерфейсом и повсеместно используется в инфраструктуре Cocoa Touch. В архитектуре MVC обязанности приложения разбиваются на отображение информации для пользователя (представление), управление нижележащими данными (модель) и реакцию на взаимодействия с пользователем (контроллер). Такое деление упрощает логику приложения и позволяет повторно использовать большие фрагменты кода независимо от конкретного устройства (то есть как на iPhone, так и на iPad).



MVC — стандартный паттерн проектирования, который вовсе не ограничивается разработкой iOS. Например, он широко применяется в Ruby on Rails.

★ КТО И ЧТО ДЕЛАЕТ? ★

Соедините каждый компонент MVC с функцией, которую он выполняет в приложении iOS.

Компонент

Описание

Модель

Обеспечивает взаимодействие с пользователем. Содержит кнопки, графику, текст и т. д. — короче, все, что видит пользователь в любой конкретный момент времени.

Представление

Отвечает за данные. Информация может храниться локально или в облаке, с использованием списков plist, баз данных, веб-служб и т. д.

Контроллер

Отвечает за координацию ответов на действия пользователя или события, происходящие в приложении. Своего рода «регулятор», управляющий информационными потоками в приложении.

КТО И ЧТО ДЕЛАЕТ? РЕШЕНИЕ

Соедините каждый компонент MVC с функцией, которую он выполняет в приложении iOS.

Компонент

Описание

Модель

Обеспечивает взаимодействие с пользователем. Содержит кнопки, графику, текст и т. д. — короче, все, что видит пользователь в любой конкретный момент времени.

Представление

Отвечает за данные. Информация может храниться локально или в облаке, с использованием списков *plist*, баз данных, веб-служб и т. д.

Контроллер

Отвечает за координацию ответов на действия пользователя или события, происходящие в приложении. Своего рода «регулирующий», управляющий информационными потоками в приложении.

Часто Задаваемые Вопросы

В: В каждом приложении iOS есть модель, представление и контроллер?

О: Да и нет. В каждом приложении наверняка есть одно представление и соответствующий контроллер представления. В простых приложениях модель, обеспечивающая хранение данных, может отсутствовать.

В: Я не вижу в коде InstaTwit ничего похожего на модель. Она там есть?

О: Одна из трудностей с кодом iOS и многих примеров заключается в том, что обязанности модели часто возлагаются на контроллер. В итоге появляется огромный контроллер представления, который пытается делать «все сразу». Сопровождение такого кода превращается в сущий кошмар. А вы видите, где в InstaTwit проявляется эта проблема?

В: Нет...

О: В контроллере представления InstaTwit содержатся массивы *feelings* и *activities*. Мы выбрали такое решение сознательно, чтобы код первого проекта был покороче, но в более масштабном проекте это создало бы проблемы. В наших последующих проектах данные будут выделяться в модель. Помните — мы вас предупредили.



Послушайте, мне нужна помощь. Я хочу вывести свой ресторан в эти, как их... социальные сети, но у меня на работе нет даже компьютера, только iPhone. Помогите!

Марко нужно мобильное приложение для социальных сетей.

К счастью, в iOS работать с Twitter и Facebook несложно. Считается, что любой ресторан должен иметь свой сайт и присутствовать в социальных сетях. Не секрет, что у шеф-повара дел по горло, и ему некогда возиться с компьютером. Посмотрим, что же Марко хочет от своего приложения.

Марко, шеф-повар и владелец ресторана. С техникой не в ладах.

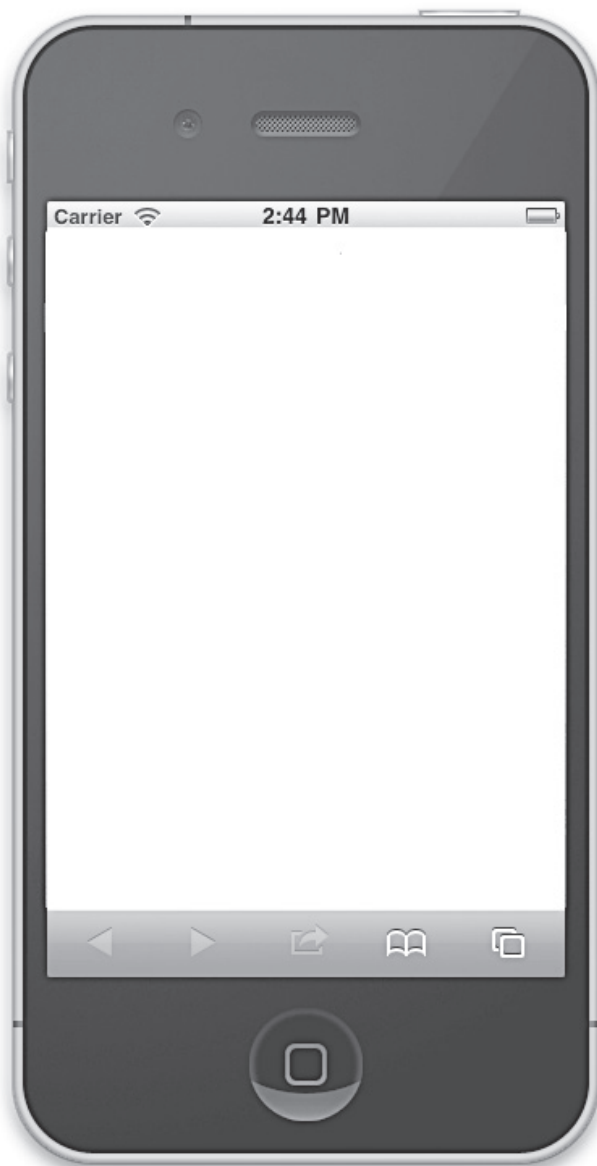
Что мне нужно:

1. Простая отправка твитов.
2. Простое обновление моей страницы в Facebook.
3. Во всех твитах должен присутствовать тег #marcos.

Возьми в руку карандаш



Вы в общих чертах представляете, как должно работать приложение. Мы ограничимся одним представлением; оно должно обновлять свое состояние и поддерживать публикации в Twitter и Facebook. Сделайте набросок, как оно, по-вашему, должно выглядеть.



Это очень важная часть разработки iOS. Не жалейте времени на построение набросков и эскизов приложений, которые вы собираетесь создать. Код всегда пишется во вторую очередь.

Начинаем с Xcode и Git

Сейчас вы становитесь на путь профессиональной разработки iOS, поэтому новый проект будет создаваться с включенной поддержкой Git. Сначала мы создадим пустой проект, а затем разберемся с дизайном приложения.

1 Создайте новый проект в Xcode.

При запуске Xcode открывается заставка; найдите команду «Create a new Xcode project» (слева в центре). Если среда Xcode уже запущена, выберите команду File→New→Project...

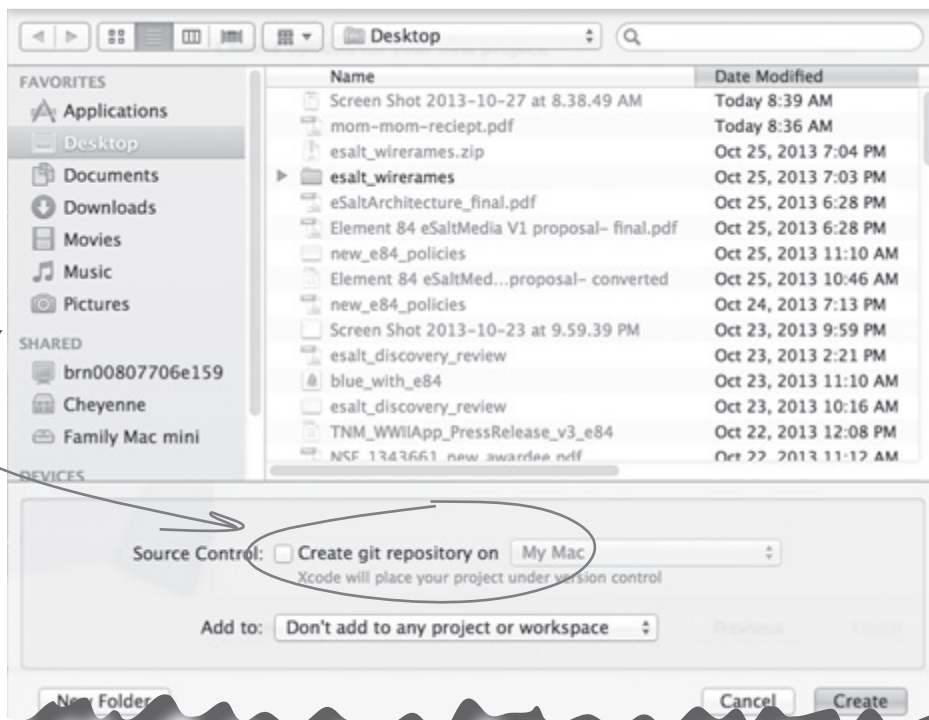
2 Выберите вариант Single View Application.

Щелкните на кнопке Next.

3 Задайте параметры нового проекта.

Введите имя приложения «MarcoPollo»; оставьте префикс класса (Class Prefix) по умолчанию. Выберите целевое устройство iPhone и щелкните на кнопке Next.

4 Выберите каталог для сохранения приложения. Проследите за тем, чтобы флажок Source Control был установлен, а справа был выбран вариант «Create git repository on My Mac». Нажмите кнопку Create.



Возьми в руку карандаш

Решение

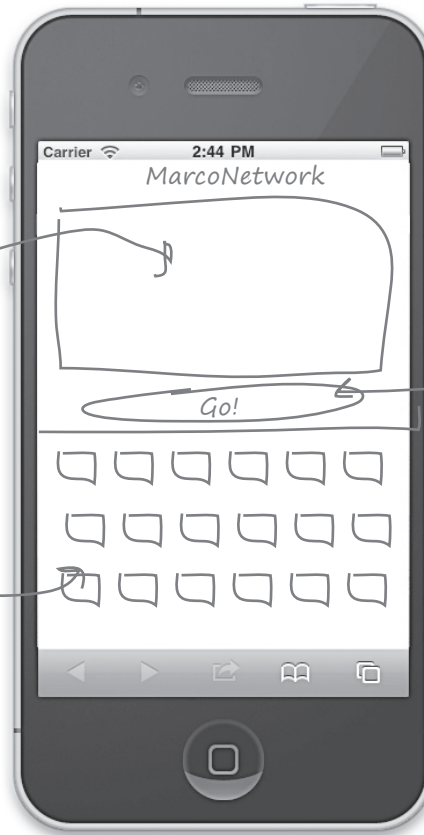


Вот какой дизайн приложения Марко получился у нас. В нем предусмотрено место для ввода информации состояния и средства публикации этой информации.

В этой области Марко будет вводить свои сообщения.

Это клавиатура. Так как приложение предназначено для отправки сообщений, она будет видна все время.

Кнопка для отправки сообщений в Facebook и Twitter.



Часто задаваемые вопросы

В: А у меня получился другой дизайн. Это нормально?

О: Конечно! Существует много подходов к проектированию приложений. Если вы обратитесь к похожим приложениям в App Store (Hootsuite, Seezmic, даже Facebook), вы найдете в них похожий дизайн. Полезно познакомиться с приложениями, решающими аналогичные задачи, чтобы получить представление об удачных дизайнерских решениях.

В: Работу над приложением следует начинать с проработки дизайна?

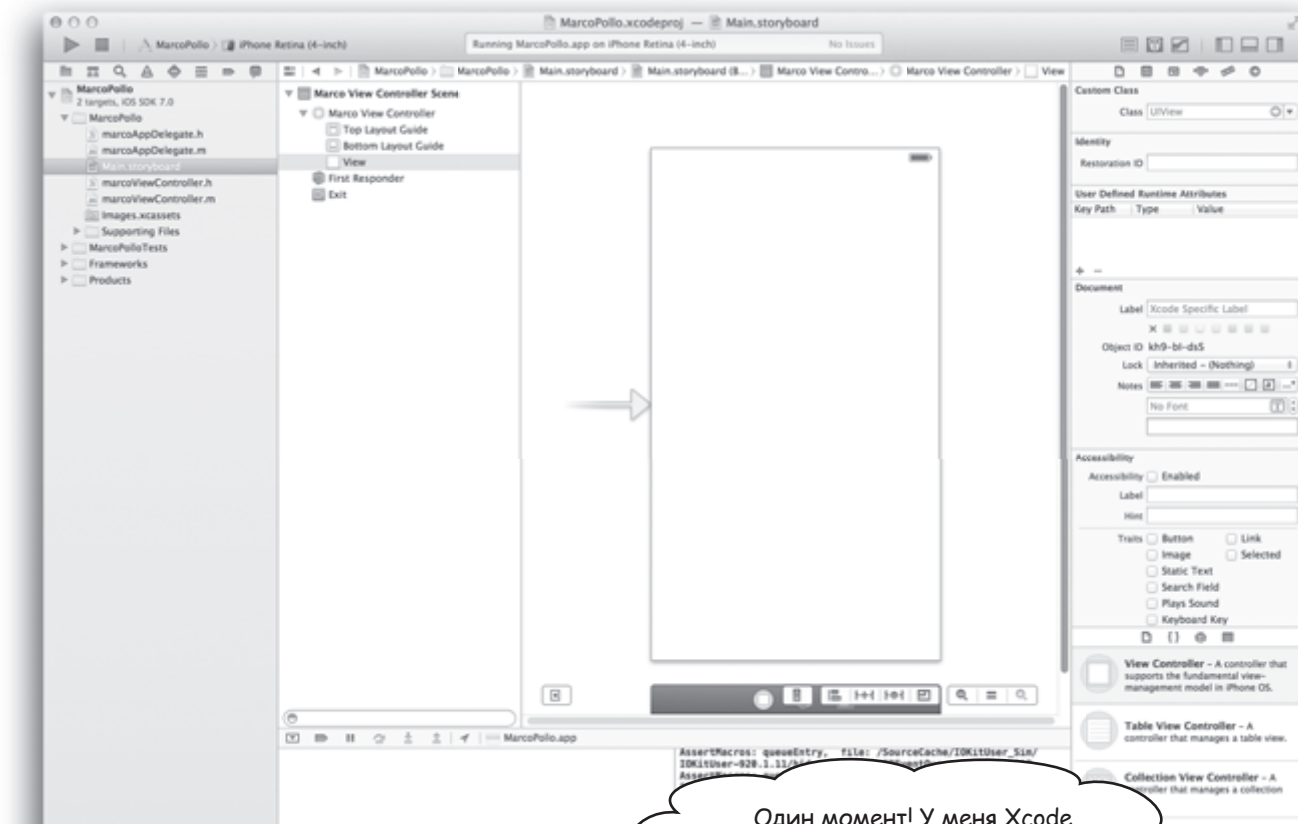
О: Взаимодействие с пользователем на небольшом мобильном устройстве играет очень важную роль. Сделать все правильно с первого раза очень трудно. С получением обратной связи от пользователей и добавления новых возможностей дизайн будет развиваться. Оставьте то, что необходимо, избавьтесь от всего остального, а дальше стройте приложение от этой отправной точки.

В: Стоит ли ограничить длину текста в текстовом поле?

О: Да! Длина сообщений Twitter ограничивается 140 символами. Соответствующую проверку мы добавим позднее, но отметить этот момент на стадии дизайна определенно стоит.

Переходим к построению интерфейса!

Пора браться за Xcode и переходить к построению этого представления. В Xcode есть удобный редактор для построения графических интерфейсов (GUI); чтобы отредактировать представление, достаточно щелкнуть на файле *Main.storyboard*.



Один момент! У меня Xcode выглядит совсем не так. В чем дело?



У Xcode широкие возможности настройки конфигурации.

Чтобы вывести на боковую панель другие библиотеки, пощелкайте на кнопках в правом верхнем углу представления. На нижней панели отображается консольный вывод и другие сообщения.

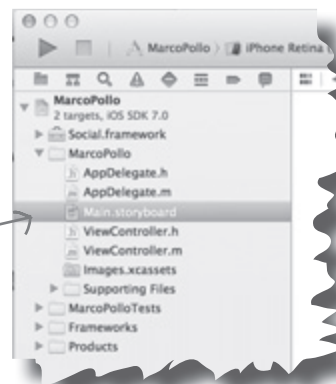
Используйте эти кнопки для настройки содержимого боковой панели. Переверните страницу, там настройка рассматривается более подробно.



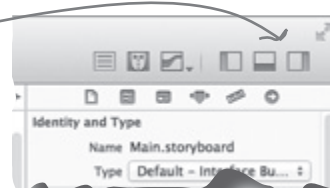


Чтобы отредактировать представления в Xcode, следует открыть файл `.storyboard` и изменить некоторые настройки рабочего пространства.

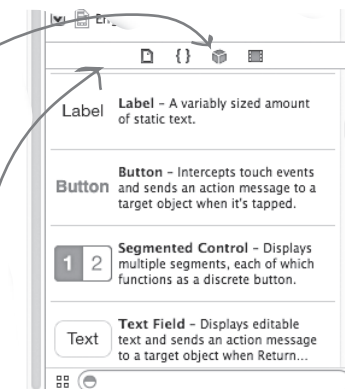
- 1 Выделите файл `Main.storyboard`.



- 2 Вызовите содержимое библиотеки, открыв панель `Utilities`.



- 3 Библиотека объектов (`Objects Library`) для представлений открывается этой кнопкой.

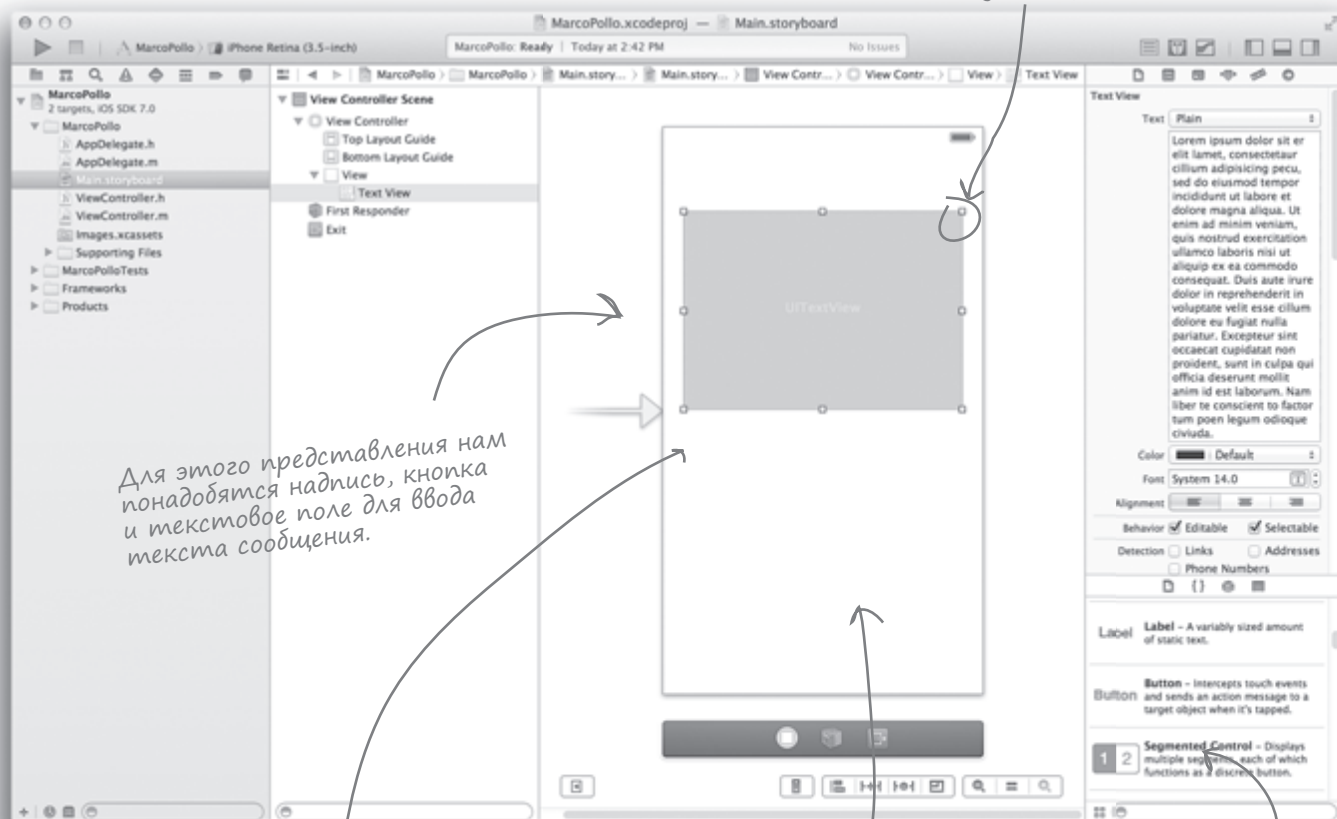


- 4 Размер панели библиотеки регулируется перетаскиванием этой полосы.

Переходим к построению интерфейса... дубль два

Раскадровки (storyboards) появились в iOS 5; эта технология упрощает редактирование пользовательского интерфейса. Интерфейс на базе перетаскивания заметно ускоряет построение макета, но в раскадровках представлены и переходы между представлениями в более сложных приложениях, включающих более одного представления. Мы уже знаем, как должно выглядеть наше приложение, поэтому работа над ним сводится к перетаскиванию и размещению нужных компонентов в нужных местах.

Скорее всего, размеры компонента, перетаскиваемого в представление, будут отличаться от желаемых. Выделите нужный компонент, захватите маленький квадратик в углу и перетяните его, пока компонент не примет нужный размер.



Для этого представления нам понадобятся надпись, кнопка и текстовое поле для ввода текста сообщения.

Эти тонкие синие линии упрощают выравнивание и настройку интервалов между элементами. Позднее некоторые из этих возможностей будут рассмотрены более подробно; а когда нужно будет обеспечить поддержку iPhone 5 и iPhone 4, мы воспользуемся относительным распределением.

В этой области будет размещаться клавиатура. За отображение клавиатуры отвечает iOS, так что от вас потребуется только выделить место под нее.

Найдите в списке кнопку (Button), текстовое поле (TextView) и надпись (Label) и перетяните их в представление.

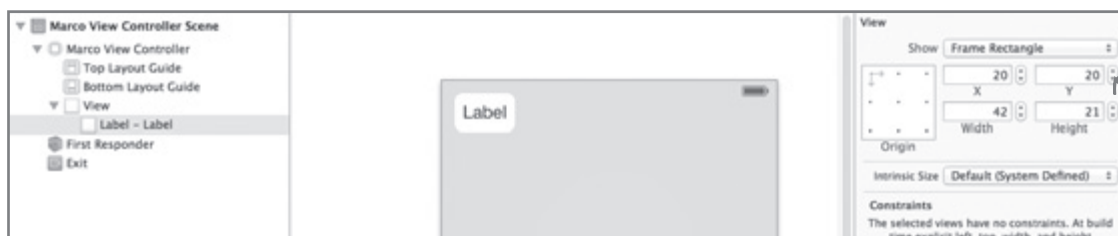
За дело!



Построение раскладки

Сейчас мы займемся добавлением и настройкой всех компонентов представления. Начнем с надписи (UILabel) в верхней части представления.

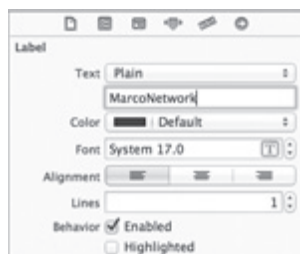
Захватите надпись и перетащите ее в левый верхний угол представления. Во время перетаскивания под контроллером представления появляются ограничения. Они выбираются на основании рекомендаций Human Interface Guidelines (HIG) компании Apple; как правило, вам стоит соблюдать эти рекомендации и оставить предлагаемое пространство.



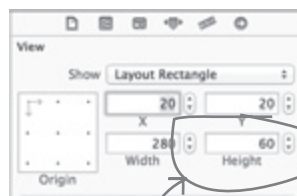
Оба ограничения должны отображаться, если в инспекторе установлен флажок «standard».

Измените текст надписи и исправьте высоту. Все эти операции можно выполнить на панели инспектора атрибутов (Attributes Inspector) в правой части редактора.

В инспекторе введите новый текст надписи *MarcoNetwork*.



Выберите панель *Size* на панели инспектора атрибутов...



И введите новую высоту 60.



Построение раскладки (продолжение)

Аналогичные изменения необходимо внести в конфигурацию двух других элементов представления, UITextView и UIButton.



Наконец, добавьте в нижней части представления кнопку, соблюдая предложенное по умолчанию расстояние от краев.



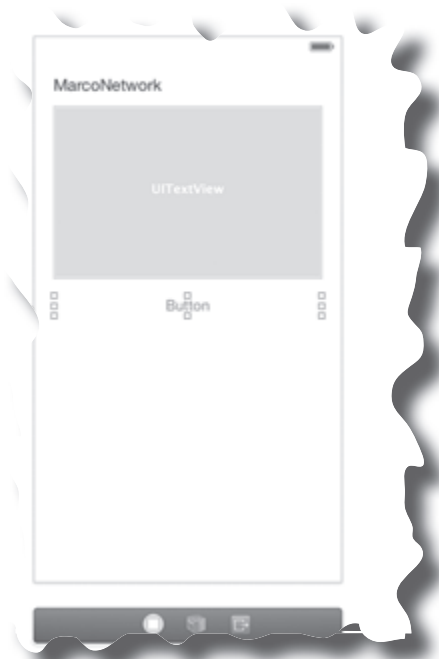


ТЕСТ-ДРАЙВ

1 Это ваша раскадровка в Xcode.

В данный момент представление должно содержать три элемента: надпись в верхней части экрана, текстовое поле для сообщения и кнопку для выполнения операции.

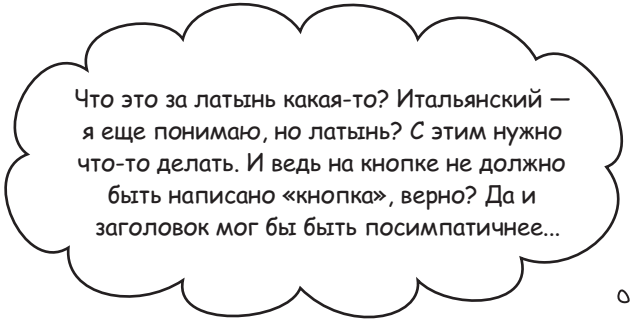
Мы воспользовались Interface Builder для настройки макета и изменения высоты некоторых элементов, чтобы привести представление к нужному виду.



2 Щелкните на кнопке запуска. Посмотрите, что получилось!

Внешний вид приложения соответствует описанию из файла *.storyboard*.





Стандартные элементы управления такие... стандартные.

В изначальном состоянии готовые элементы управления не впечатляют. Впрочем, вы их можете настроить. Многие элементы управления поддерживают нестандартную графику, так что используйте в своем приложении цветовые темы, стили оформления кнопок и т. д. Будьте осторожны при настройке элементов; помните, что главное — юзабилити. И даже если вы думаете, что мигающая ядовито-зеленая кнопка выглядит стильно, пользователи могут с вами не согласиться.



НАПРЯГИ МОЗГИ

Запишите список дел для выполнения требований Марко. Сверьтесь со списком в нижней части страницы.

.....

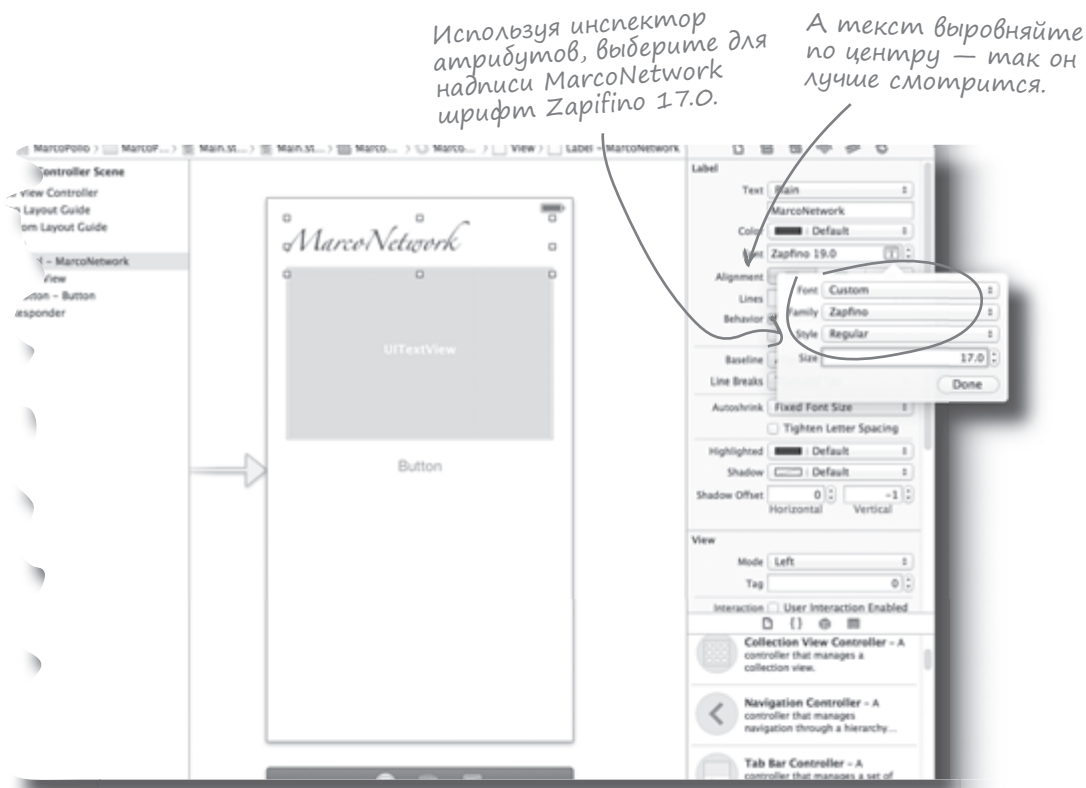
.....

.....

1. Изменить шрифт заголовка. 2. Убрать текст на панели и изменить его текст по умолчанию «your message here». 3. Заменить текст кнопки строкой «Post it!»

Xcode позволяет легко вносить косметические изменения

Мы уже пару раз вносили изменения в инспекторе, но давайте разберемся подробнее. Начнем с исправления текста по умолчанию и надписей.



Выделите кнопку и при помощи инспектора атрибутов измените текст кнопки на «Post it!»

Затем выделите текстовое поле, в инспекторе атрибутов удалите большой блок текста на латыни и введите вместо него строку «<your message here>».

В принципе то же самое можно было сделать в программном коде...

При внесении подобных изменений (размер шрифта, текст по умолчанию и т. д.) в Interface Builder среда Xcode записывает эту информацию в раскадровку, чтобы она задавалась при загрузке раскадровки приложением. Все, что здесь делается (а также многое другое), также можно сделать в коде приложения. Выбор между изменением пользовательского интерфейса в программном коде или в Interface Builder отчасти является делом вкуса — и конечно, зависит от того, предоставляет ли Interface Builder средства для внесения таких изменений.

Элемент управления iOS изнутри

Мы изменили текст на кнопке, но когда пользователь прикасается к кнопке, по-прежнему ничего не происходит. Чтобы кнопка делала что-то осмысленное, необходимо связать ее с программным кодом.



Ког кнопка вызывает действие

Если вернуться к паттерну «Модель-Представление-Контроллер» — пока все наши действия относились к представлению. Пора заняться поведением, то есть перейти к функциональности контроллера.

Чтобы связать поведение с действиями пользователя, Objective-C связывает **события** с методами действий **IBAction** (IB в имени IBAction — сокращение от Interface Builder). У каждого элемента управления имеется одно или несколько событий, которые инициируются при выполнении каких-либо действий (например, прикосновения пользователя). Событие элемента управления можно связать с блоком кода, помеченным как IBAction, в Interface Builder (это можно сделать и на программном уровне). Когда действие пользователя приводит к срабатыванию события, iOS вызывает метод действия и передает ему информацию о том, что произошло.

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after
    loading the view, typically from a nib.
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    //
    Dispose of any resources that can be
    recreated.
}

@end
```

Написанный нами код Objective-C

ViewController.m

Представление

Модель

Контроллер

Сейчас мы находимся здесь

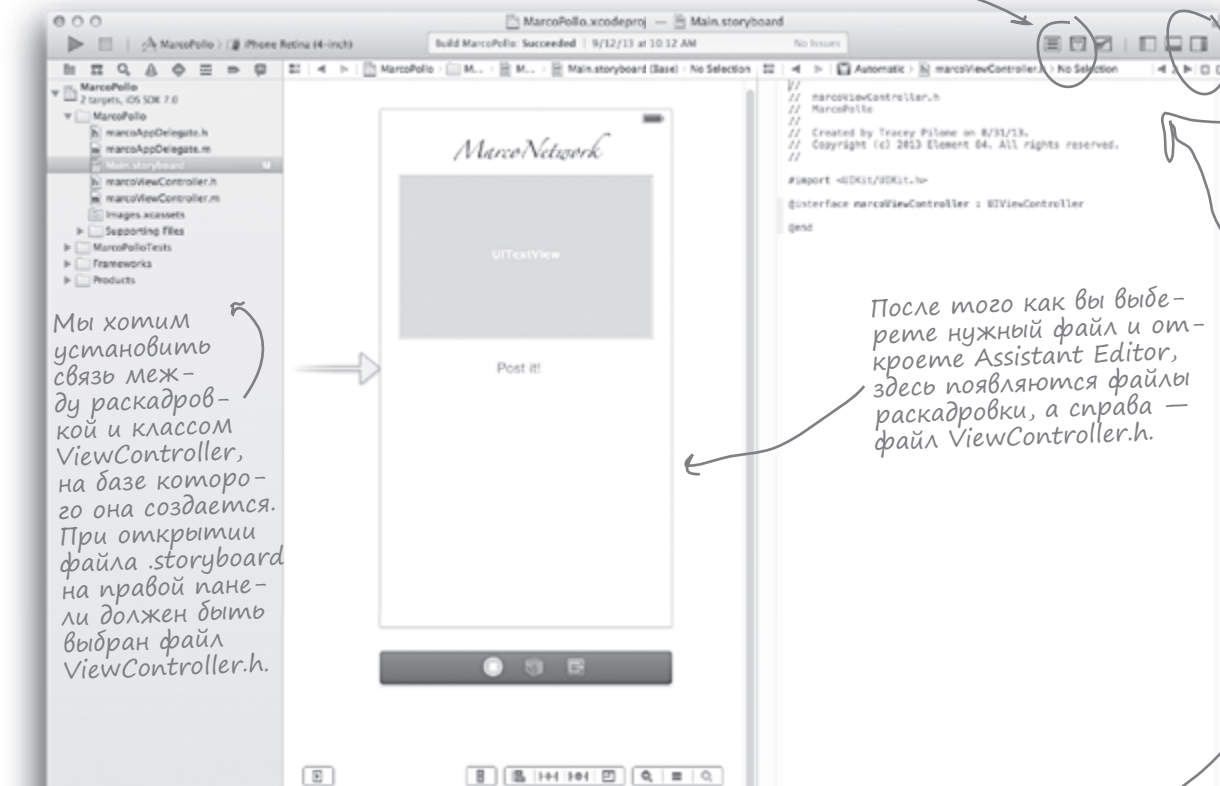


Для Любознательных

В Xcode имеется режим Assistant Editor, в котором могут отображаться обе части класса одновременно (.h и .m) или представление вместе с ассоциированным контроллером представления. В этом режиме также можно графически связывать события с IBAction. Для активизации этого режима редактирования следует закрыть правую панель в редакторе и включить режим Assistant Editor.

Откройте Assistant Editor.

Закройте правую панель.



Будьте осторожны!

Assistant Editor не всегда показывает то, что нужно.

Если на правой панели отображается файл .m, щелкните на стрелках в правом верхнем углу, чтобы вызвать на экран файл ViewController.h.

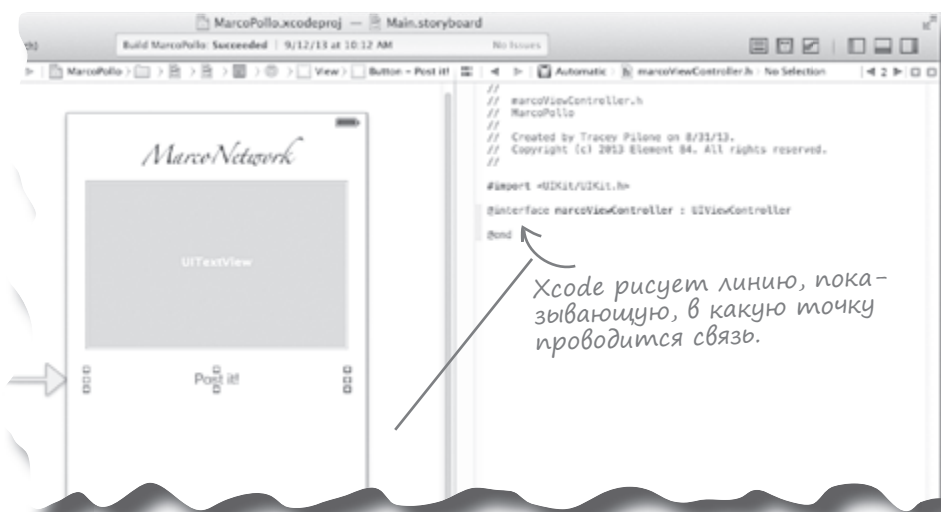
Действия создаются в редакторе графического интерфейса среды Xcode

Редактор графического интерфейса (GUI) среды Xcode позволяет визуальнo настроить действие для любого компонента (в нашем случае это кнопка «Post it!»).

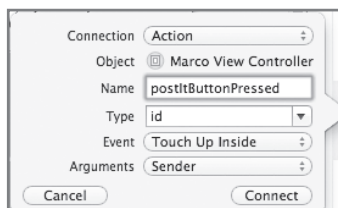
- 1 Выделите кнопку «Post it!» в раскладовке.



- 2 Щелкните с нажатой клавишей Ctrl (или используйте правую кнопку мыши) и перетащите указатель на файл `ViewController.h` на правой панели, между директивами `@interface` и `@end` в этом файле.



- 3 Настройте связь как действие (Action) с именем «postItButtonPressed». Сохраните созданную связь кнопкой Connect.



4

Xcode добавляет в заголовочный файл объявление действия, а в файле реализации создается пустая реализация. Просто чтобы убедиться в том, что все прошло правильно, включим в реализацию `.m` команду вывода сообщения. Программный код выглядит так:

Найдите метод действия (IBAction), сгенерированный в файле `.m`.

```
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

(IBAction)postItButtonPressed: (id) sender {
    NSLog(@"Post It button was pressed!");
}

@end
```

Поместите код вывода сообщения в фигурные скобки.



ViewController.m



ТЕСТ-ДРАЙВ

Post It!

Одну минуту... Я понятия не имею, что здесь происходит. Да, мы создали действие... А дальше-то что?

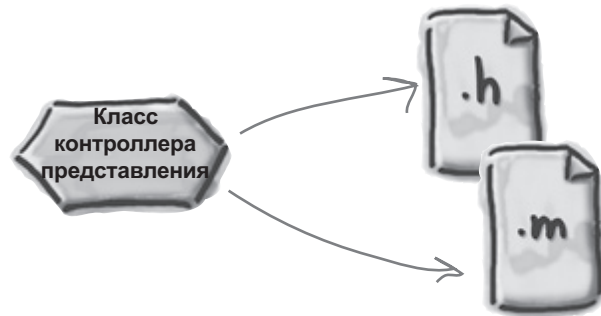


Связывание элементов управления с действиями

Это ваш элемент управления



Это ваш контроллер...



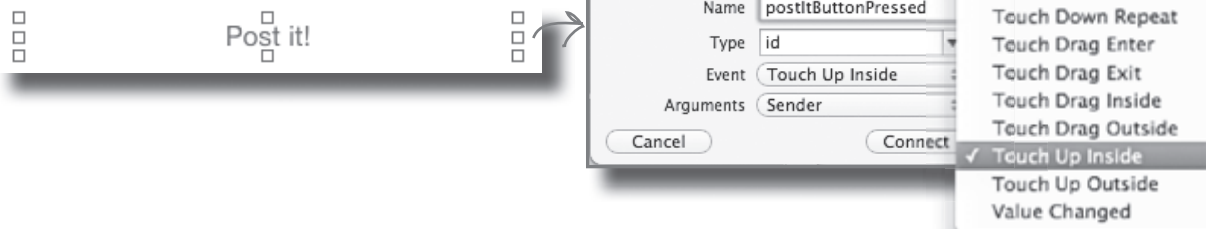
Элемент управления представляет визуальный компонент, содержащий информацию о шрифте, цвете текста, графике нажатого и ненажатого состояния, цветах фона и т. д. Он отвечает за отображение информации и ввод данных. Элементы управления могут использоваться в разных местах, поэтому неразумно включать в них код, привязанный к конкретному приложению. Элементы управления просто инициируют события, сообщая окружающим: «Эй! Тут у меня кое-что произошло...»

Логика приложения реализуется в коде, который вы пишете для своих приложений. Она принципиально отделяется от визуальных аспектов — того, что видит пользователь. Программисту не нужно писать код перерисовки кнопок в каждом приложении, которое он создает. Он работает над поведением — тем, что должно делать приложение «на высоком уровне» (например, отправлять сообщения для Марко).

У элемента управления есть события...

При настройке связи в диалоговом окне Xcode было выбрано событие по умолчанию. Если вернуться к диалоговому окну и щелкнуть на кнопке со стрелкой, вы увидите список событий, которые могут инициироваться приложением.

Все варианты выводятся только для нового события (а не того, которое уже было настроено).

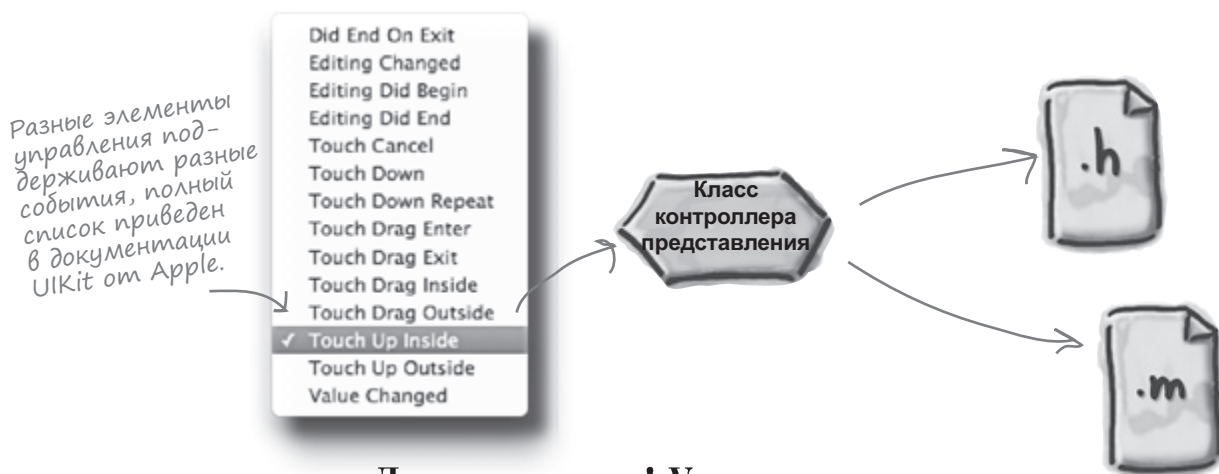


...и эти события можно связывать с действиями

Разные элементы управления могут инициировать разные события, но в общем случае событие сообщает системе о том, что пользователь что-то сделал. Например, мы связали событие «Touch Up Inside» с нашим действием, чтобы узнать, когда пользователь отведет палец от экрана внутри кнопки. Обычно в приложениях не задействуется только событие Touch Up Inside.

При связывании события с действием iOS передает информацию о том, какой элемент управления отправил событие; это означает, что при желании одно и то же действие может использоваться для обработки разных событий от разных элементов управления (нужно лишь определить, кто именно отправил событие). С другой стороны, код усложняется по сравнению с решениями, использующими разные методы действий, поэтому чаще в программах создается отдельное действие для каждого события, на которое должна реагировать программа.

Устанавливая связи событий с действиями, вы связываете операции, выполняемые пользователем, с кодом приложения.



Дело сдвинулось! У нас есть элемент управления, событие и действие. Но нам предстоит еще многое сделать...

- ❶ Получить текст, который ввел Марко.
- ❷ Добавить в него рекламную часть.
- ❸ Опубликовать сообщение Марко.
- ❹ Получить бесплатный обед!

Не огорчайтесь, если какие-то термины вам непонятны! Через несколько страниц вы все узнаете, но попробуйте выполнить упражнение сейчас!

КТО И ЧТО ДЕЛАЕТ?

Соедините каждый компонент разработки для iOS с его описанием.

Компонент

Описание

Элемент управления

Визуальный элемент, который может реагировать на операции пользователя и инициировать события для любой стороны, которая пожелает эти события обрабатывать.

IBAction

Объект, предоставляющий логику работы представлений iOS. Как правило, содержит ссылки на элементы представления, код реакции на операции пользователя, а также логику перехода на другие представления.

IBOutlet

Событие, которое инициируется кнопкой для обозначения того, что пользователь оторвал палец от экрана внутри области кнопки, а проще говоря, нажал кнопку.

Контроллер представления

Индикатор свойства класса, который сообщает Interface Builder, что это свойство связывается с элементом пользовательского интерфейса.

Touch Up Inside

Код (Objective-C), вызываемый при инициировании соответствующего события.

Часть Задаваемые Вопросы

В: Мне встретился термин «файлы nib». Что это такое?

О: До появления файлов раскадровки storyboard использовались файлы nib, которые в действительности имели расширение .xib. Теперь эти файлы описывают одно отдельное представление в составе файлов storyboard. При желании вы можете работать и с файлами nib, но в контексте книги лучше придерживаться практики, рекомендованной Apple.

В: Что собой представляют файлы storyboard?

О: Они содержат данные в формате XML. Это информация о том, как должны отображаться представления и как осуществляются переходы между ними.

По мере становления дисциплины разработки iOS стало очевидно, что в большинстве приложений используется несколько представлений, а переключение между этими представлениями создает все больше трудностей для разработчиков. Раскадровка помогает рассматривать приложение как единое целое и понять, как ваши представления работают в сочетании друг с другом.

В: Это уже второе наше приложение для iPhone, как насчет других устройств?

О: И до них дойдет! Пока мы стараемся не усложнять материал. Кроме того, это отличная возможность задуматься об удобстве использования. Существует немало приложений, для которых имеет смысл только реализация на iPhone. Приложения для бегунов, приложения для путешественников — словом, любые приложения, которые вряд ли кому-нибудь захочется использовать на iPad.

В: А если я захочу добавить поддержку iPad позднее?

О: Хороший вопрос. Это можно сделать двумя основными способами. Прежде всего, можно построить универсальное приложение вместо приложения, предназначенного только для iPhone. Кроме того, в наших представлениях учитывается еще одно обстоятельство: вместо фиксированных состояний в дизайне наших представлений мы определяем интервальные распределения.

Разработка для iOS имеет много общего с веб-программированием и даже (о ужас!) программированием для Android. С выходом

iPad mini и различных экранов Retina придется поддерживать много разных размеров устройств, причем в будущем этот набор будет только расширяться. Один из способов сохранить жизнеспособность вашего приложения — строить представления на основе интервалов вместо фиксированных позиций компонентов.

В: Мне не нравятся редакторы GUI, я предпочитаю работать в командной строке. Что делать?

О: Это тоже возможно. Впрочем, мы бы не рекомендовали этот путь. Компания Apple потратила много времени и сил на то, чтобы усовершенствовать свой редактор GUI и сделать его удобным для разработчиков. Попробуйте поработать с ним — хотя бы недолго!

В: Вы начали говорить о действиях. Но ведь пока о синтаксисе ничего толком не сказано...

О: Вскоре мы доберемся и до синтаксиса, честное слово! Просто продолжайте читать...

★ КТО И ЧТО ДЕЛАЕТ? ★ РЕШЕНИЕ

Соедините каждый компонент разработки для iOS с его описанием.



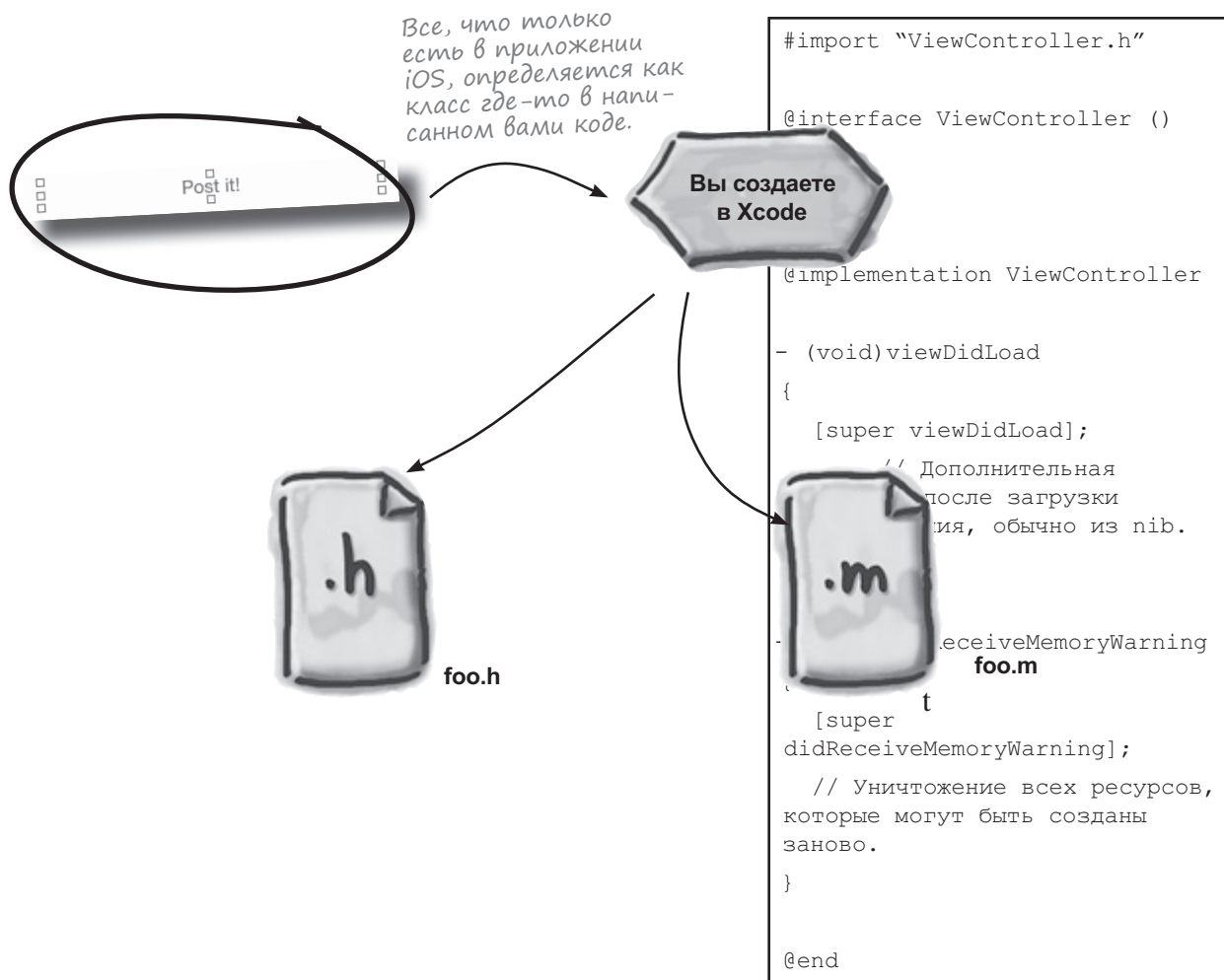
Итак, о классах и интерфейсах...

Мы снова и снова упоминаем контроллеры представлений и пользовательские интерфейсы, но так толком и не объяснили, как же работают классы и интерфейсы. Мы добавили в интерфейс контроллера представления метод (IBAction)... и все. Давайте выделим пару минут и познакомимся с классами поближе. В конце концов, всю оставшуюся часть книги мы будем заниматься написанием классов, так что лучше заранее понять, с чем же мы будем иметь дело...

Классы... под увеличительным стеклом



Objective-C: классы на каждом шагу



Интерфейс — то, что делает ваш класс.

Интерфейс объявляет, что может сделать ваш класс и как он должен использоваться другими объектами.

Реализация — то, как ваш класс это делает.

Файл реализации описывает, как ваш класс делает то, что он должен делать в соответствии с заявленным интерфейсом. Другие объекты не интересуются, как реализована та или иная возможность, главное — они знают, что вы можете это сделать!

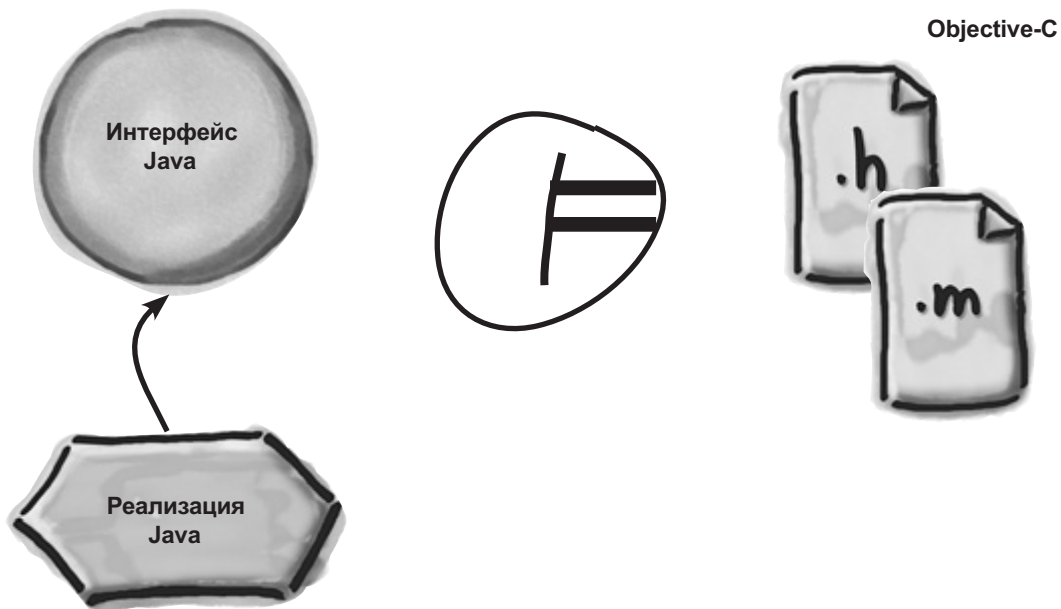
Написанный код делится на классы. Класс имеет интерфейс и реализацию.



Стоп, я уже имел дело с интерфейсами.
Это как в Java, верно? Класс реализует
интерфейс.

Не совсем...

В Objective-C файл `.h` просто объявляет открытый (общедоступный) интерфейс класса, включая все свойства, методы, и возможно — приватные переменные (хотя обычно они перемещаются в файл `.m` и не входят в открытый интерфейс. Интерфейсы Java часто сравнивают с протоколами Objective-C, о которых мы поговорим позже.



Часто Задаваемые Вопросы

В: А во всех этих событиях трудно разобраться?

О: Существует множество событий, на которые iOS реагирует для разных элементов управления; и кнопка является одним из самых простых случаев. Слова в текстовом описании события тщательно выбираются. Например, «Touch up inside» («Отпускание пальца внутри») используется потому, что iOS должна реагировать на окончание прикосновения к кнопке, а не на его начало.

За дополнительной информацией о событиях и описаниями обращайтесь к руководству «Event Handling Guide» в документации разработчика iOS.

В: Но я вижу ошибки компиляции?

О: Бывает. Xcode обычно выдает полезные предупреждения перед запуском приложения. Индикаторы обычно выводятся в верхней части окна. Если вы видите маленькие красные восклицательные знаки или желтые треугольники, значит, что-то пошло не так.

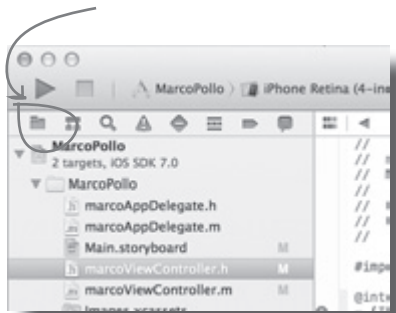


Вы можете щелкнуть на них, чтобы перейти прямо к проблемному коду. А оказавшись на нужном месте, вы узнаете, что же не нравится Xcode.

В: Как скрыть или вызвать на экран отладчик?

О: Если приложение останавливается во время выполнения в эмуляторе, происходит возврат прямо в Xcode в представлении отладчика.

Чтобы вернуться к нормальному представлению с файлами, щелкните здесь:



В: Что это за @ в начале строк?

О: Не забывайте, что Objective-C является объектно-ориентированным языком. Обычно для работы со строками в Objective-C используется класс NSString, но из-за частого использования строк в Objective-C поддерживается сокращенное обозначение инициализации объектов NSString: знак '@'.

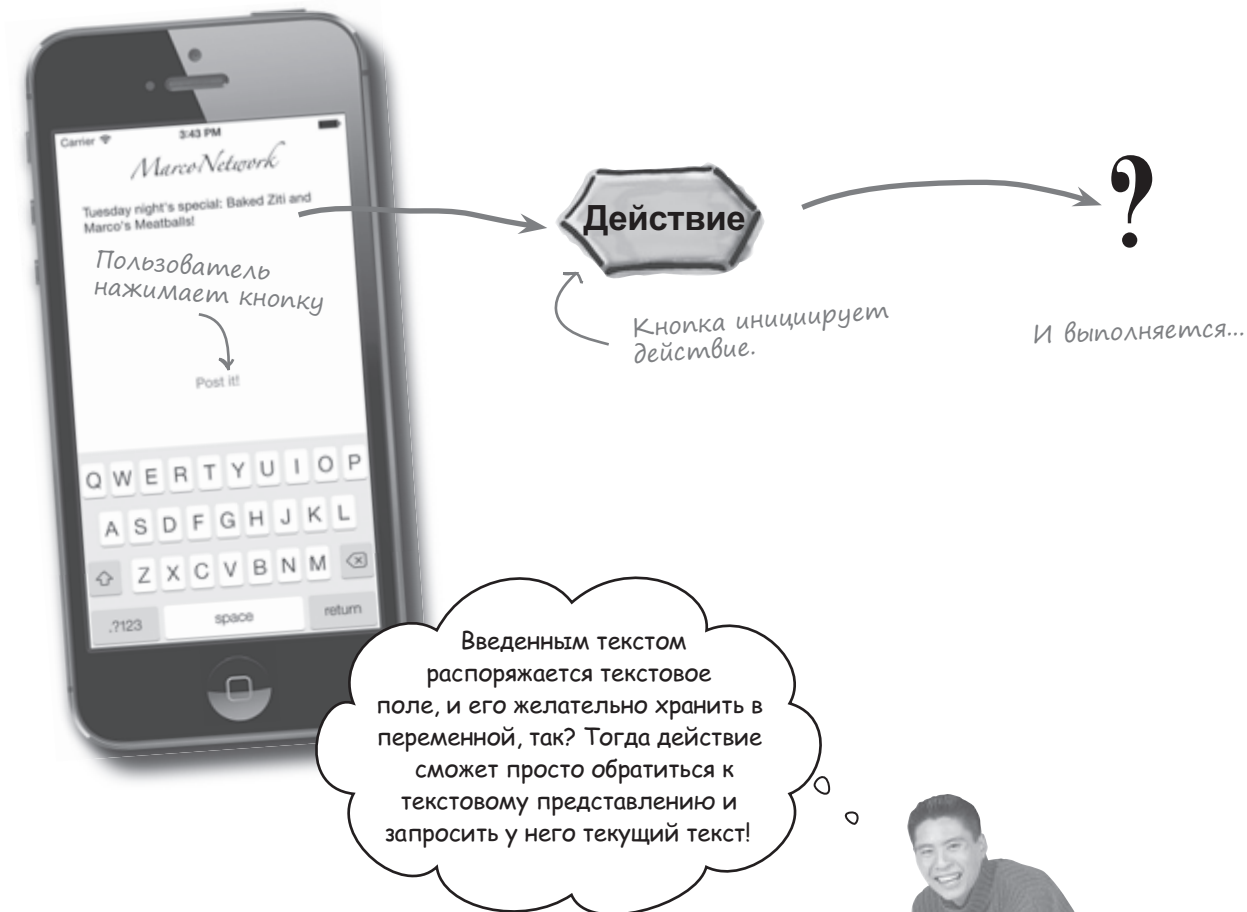


Все это замечательно... но как
поживает мое приложение?
Я хочу, чтобы в мой ресторан
приходили клиенты!



И как добраться до текста?

Итак, у нас имеется действие, вызываемое при нажатии кнопки. Теперь нужно как-то получить доступ к тексту сообщения, чтобы опубликовать его в Twitter!



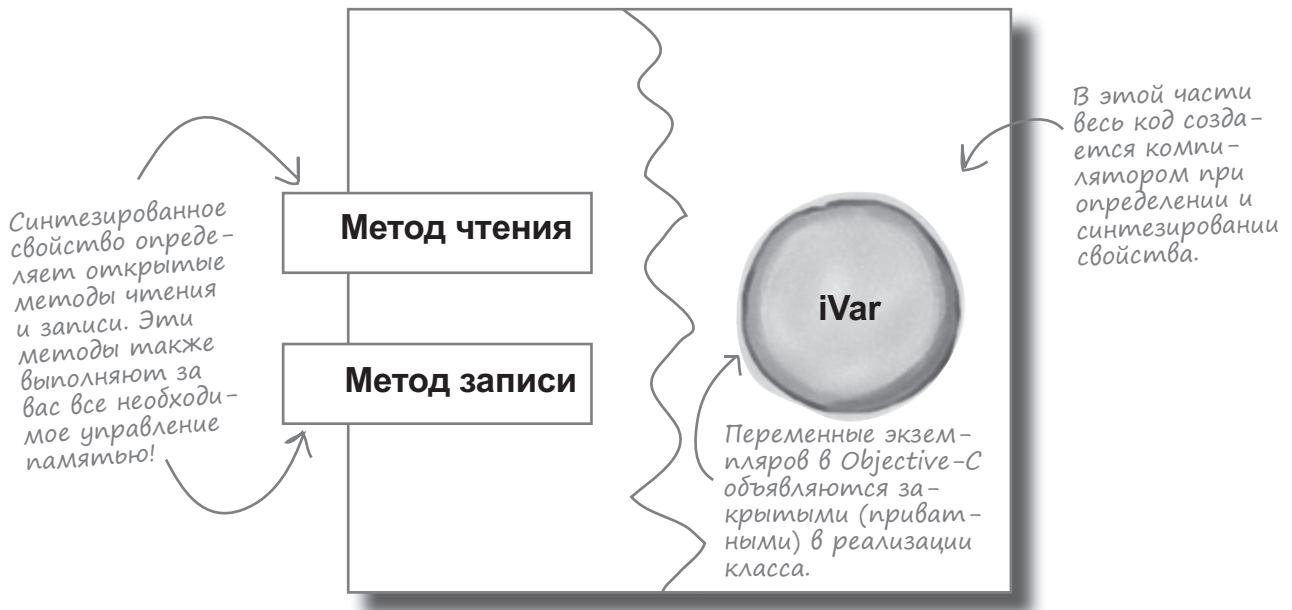
Правильно! В Objective-C для обращения к переменным используются свойства.

В отличие от других языков, в которых методы доступа приходится создавать явно, свойства Objective-C поручают всю «черную работу» компилятору. Вам остается лишь определить свойство в файле `.h`.

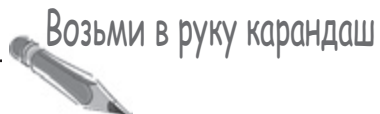


Методы чтения и записи свойств создаются автоматически

Директива `@property` в заголовочном файле сообщает компилятору об определении свойства. В современном языке Objective-C методы доступа свойств синтезируются автоматически — включать директиву `@synthesize` в файл реализации не обязательно. В синтезированных свойствах компилятор генерирует метод чтения, а для свойств с возможностью чтения/записи — метод записи и реализует их на основании атрибутов `@property`, объявленных в файле `.h`.



Как же все
это выглядит
в Objective-C?



Из приведенных ниже фрагментов кода только один сохраняет текстовое поле для публикации сообщения. Обведите этот фрагмент кружком!

```
NSLog(@"Post It button  
was pressed: %@", self.tweetTextView.text);
```

```
@property (weak, nonatomic) IBOutlet UITextView *tweetTextView;
```

```
self.tweetTextView.  
font=[UIFont fontWithName:@"Helvetica" size:15.f];
```

```
NSString *buttonTitle=[sender titleForState:UIControlStateNormal];
```

Создайте свойство для текстового поля

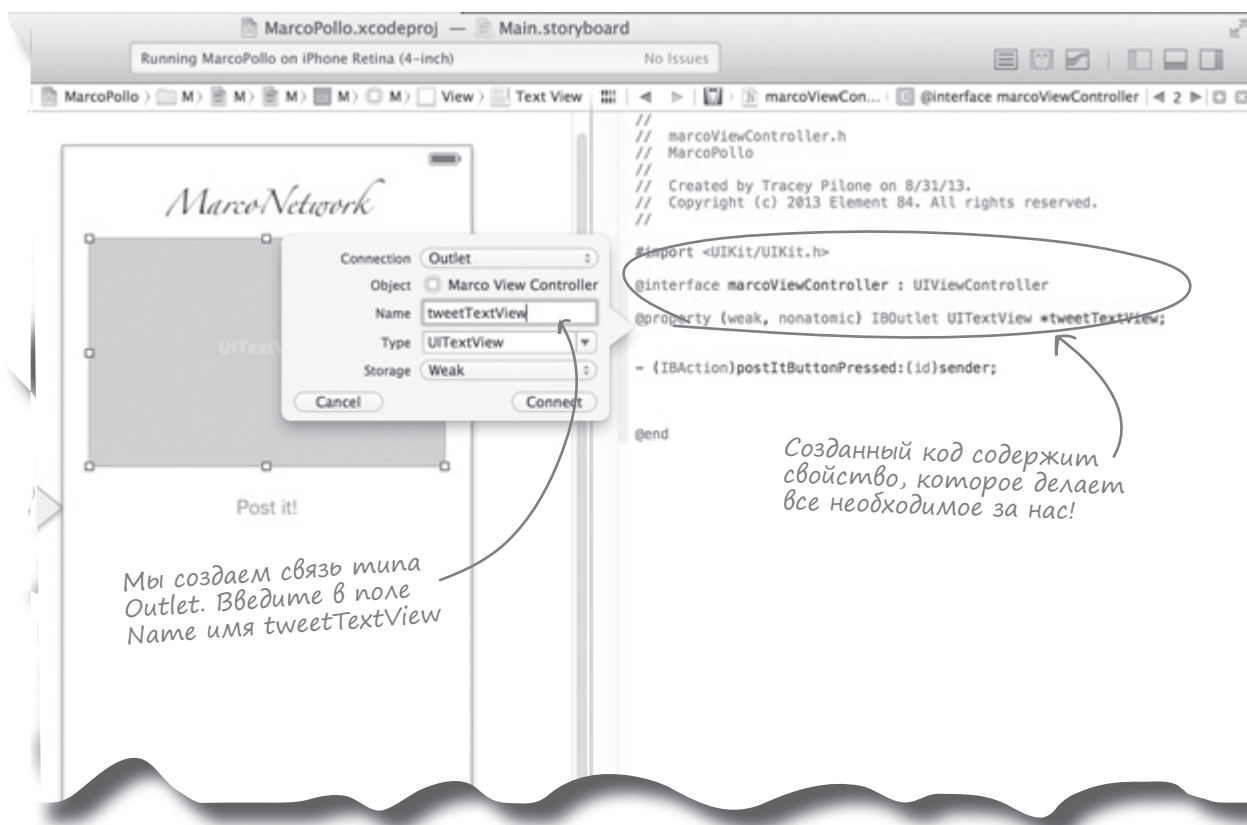
Нам понадобится свойство для упаковки всего, что относится к текстовому полю с сообщением, а также механизм получения информации. В редакторе раскадровки Storyboard Editor мы снова создадим все необходимое для подготовки текста и использования его в приложении.



Упражнение!



Выделите текст, нажмите клавишу Control и перетащите указатель мыши между директивами @interface и @end в файле *ViewController.h*.



РАССЛАБЬТЕСЬ

Синтаксис свойств будет более подробно описан ниже.

Возьми в руку карандаш



Решение

Мы предложили несколько вариантов для выбора фрагмента кода, используемого для сообщения. А какой вариант выбрали вы?

```
NSLog(@"Post It button  
was pressed: %@", self.tweetTextView.text);
```

Использует тексто-
вое поле для вывода
сообщения на консоль.

```
@property (weak, nonatomic) IBOutlet UITextView *tweetTextView;
```

А вот и нужный код. Мы
создаем свойство IBOutlet
и связываем его с текстовым
полем в раскладовке.

```
self.tweetTextView.  
font=[UIFont fontWithName:@"Helvetica" size:15.f];
```

Изменяет шрифт
текстового поля
с сообщением.

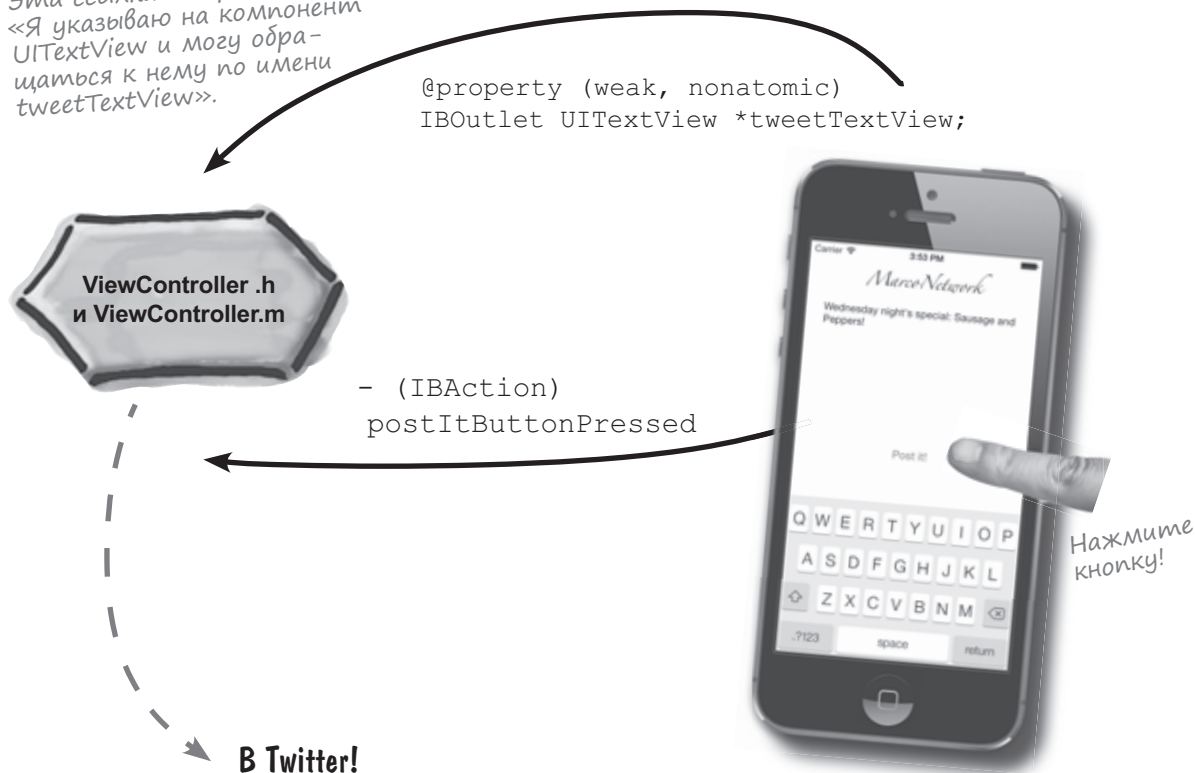
```
NSString *buttonTitle=[sender titleForState:UIControlStateNormal];
```

Отправителем ('sender') в данном случае
является кнопка с текстом «Post it».
Этот код просто получает текст кнопки
в нормальном состоянии.

Элементы управления связываются со ссылками IBOutlet

Создание ссылки IBOutlet было последним шагом, необходимыми для взаимодействия с текстовым полем. Так как в текстовом поле хранится текст сообщения, нам нужен какой-то способ для обращения к нему из программного кода. В данном случае мы работаем с элементом управления из пользовательского интерфейса, а не просто реагируем на событие.

Эта ссылка говорит:
«Я указываю на компонент
UITextView и могу обра-
щаться к нему по имени
tweetTextView».



Ссылка IBOutlet относится к компоненту пользовательского интерфейса

Так как пользователь взаимодействует с интерфейсом, нам необходим способ обращения к этому интерфейсу и его обновления в программном коде. Для отправки сообщений нужно реагировать на нажатия кнопок (IBAction) и получить доступ к тексту, введенному пользователем (IBOutlet).



ТЕСТ-ДРАЙВ

1 Создайте связь со ссылкой IBOutlet.

Если это не было сделано ранее, вернитесь назад и перетащите в раскадровке указатель мыши с нажатой клавишей Ctrl в файл *ViewController.h*.

2 Выведите сообщение на консоль.

После создания ссылки IBOutlet все готово — нужно только как-то проверить, что ссылка работает. Отредактируйте сообщение NSLog в методе *postItButtonPressed* в файле *ViewController.m*.

```
- (IBAction)postItButtonPressed:(id)sender {
    NSLog(@"Post It button was pressed: %@", self.tweetTextView.
text);
}
```



ViewController.m

3 Запустите приложение в эмуляторе.

Постройте и запустите приложение. Когда оно заработает, введите на месте текста <your tweet here> что-то более содержательное и нажмите кнопку «Post It»...



Часто задаваемые вопросы

В: Напомните, что такое событие?

О: Элементы управления генерируют события, когда с ними что-то происходит. Вы можете связать эти события с методами, чтобы при возникновении события вызывался соответствующий метод. Как правило, события связываются с методами в Interface Builder, но также можно создавать такие связи в программном коде (мы займемся этим позднее).

В: Выходит, одна строка автоматически сгенерированного кода решает сразу несколько задач?

О: Да! При создании ссылки IBOutlet (перетаскиванием указателя мыши с нажатой клавишей Ctrl) генерируется строка кода, начинающаяся с `@property`. Она объявляет переменную, создает методы чтения и записи этой переменной, а также создает ссылку для получения текста из текстового поля. И все это в одной крошечной строке!

В: Вроде раньше нужно было использовать директиву `@synthesize`?

О: Да, но это было раньше. С реализацией ARC (автоматического подсчета ссылок) компания Apple переместила большую часть операций управления памятью при

разработке для iOS из кода в компилятор. Директива `@synthesize` использовалась для создания переменной в файле реализации и запуска процесса. Теперь компилятор делает это все за вас.

В: Объявление `@property` всегда генерирует методы чтения и записи?

О: Не всегда! Свойство может быть доступным для чтения/записи или только для чтения. Когда свойство определяется как доступное только для чтения, компилятор генерирует только метод чтения. Такое определение гарантирует, что свойство не может быть изменено методом записи.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Действия реагируют на события.
- Ссылки IBOutlet предназначены для работы с компонентами пользовательского интерфейса; они используются для отображения или получения информации от пользователя.
- Редактор раскадровки способен автоматически создать большую часть кода, необходимого для создания действий и ссылок IBOutlet.
- Объявления свойств создают переменные экземпляров, а также методы чтения и записи для этих переменных.

Простая работа с Twitter

В iOS 6 впервые появились инфраструктуры для работы с Twitter и Facebook. Вместо того чтобы заставлять разработчиков разбираться с API и механизмами аутентификации на этих сайтах, Apple делает все за вас — и это замечательно, потому что технические подробности время от времени изменяются.

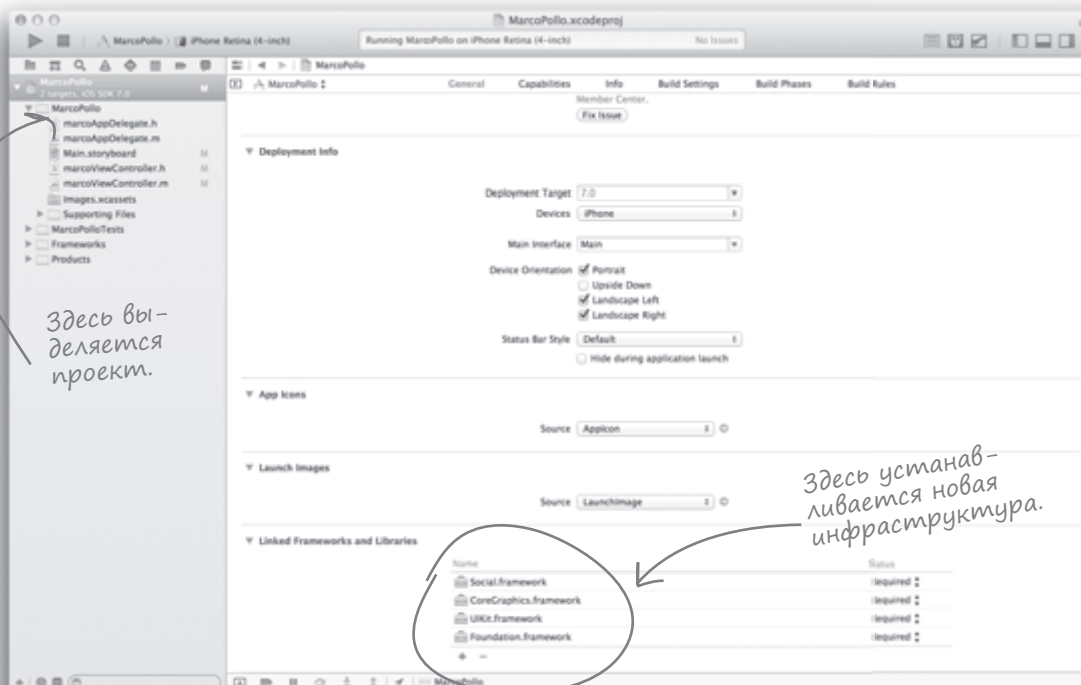
И это действительно просто — спросите знакомых разработчиков приложений Android, каково приходится им...

- ☐ В вашем проекте должна быть установлена инфраструктура для отправки сообщений (инфраструктуры — аналоги библиотек в Java).
- ☐ В приложение должен быть включен код отправки сообщения.



Готово
к употреблению

Выделите проект, убедитесь в том, что выбрана вкладка General, после чего прокрутите список до раздела Linked Frameworks and Libraries и нажмите кнопку +. Выберите в списке строку Social Framework и нажмите кнопку Add.





Готово к употреблению

Ниже приведен код, который необходимо добавить в программу, чтобы заработала поддержка Twitter. Всего несколько строк, и ваше приложение начинает отправлять сообщения в Twitter!

```
#import "ViewController.h"
#import "Social/Social.h"
```

```
- (IBAction)postItButtonPressed:(id)sender {
    NSLog(@"Post It button was pressed: %@", self.tweetTextView.text);
    SLComposeViewController *composer = [SLComposeViewController compose
    ViewControllerForServiceType:SLServiceTypeTwitter];
    [composer setInitialText:self.tweetTextView.text];
    [self presentViewController:composer animated:YES completion:nil];
}
```

Здесь вводится компоновщик — новое модальное представление, которое будет автоматически заполняться текстом сообщения Марко.



ViewController.m

Проверим...



ТЕСТ-ДРАЙВ

- 1** Проследите за тем, чтобы в проект была включена инфраструктура *Social framework* и приведенный выше код.
- 2** Постройте и запустите приложение.
Введите новое сообщение.
- 3** Настройте учетную запись *Twitter* в эмуляторе.
При нажатии кнопки «Post it!» вам будет предложено выполнить настройку учетной записи. А когда это будет сделано, публикация сообщений в *Twitter* заработает!



Отличная работа!
У дверей ресторана Марко
уже собирается очередь...

Часто
Задаваемые
Вопросы

В: Чем метод с объявлением `IBAction` отличается от остальных методов?

О: Ничем! Методы `IBAction` ничем не отличаются от любых других методов в файле реализации. Объявляя метод как `'IBAction'`, вы сообщаете IB, что он может использоваться для подключения к событиям элементов управления.

#MarcoPollo



А где мой хеш-тег? Как мне
узнать, что думают другие
люди?

**Марко хочет организовать
дополнительную обработку сообщений
перед отправкой.**

Цель Марко — эффективность и логичность.
Ну и возможно, хеш-тег #MarcoPollo пригодит-
ся для анализа и планирования. В приложение
нужно включить логику добавления хеш-тега...



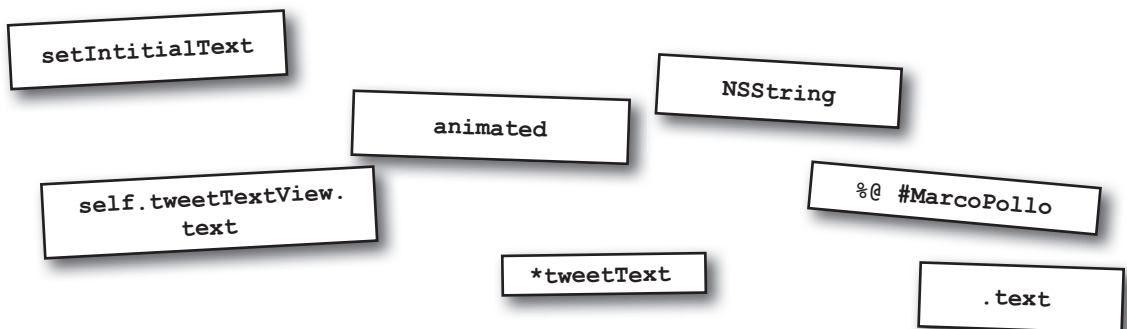
Магниты для iOS

Нужно включить в приложение код добавления хеш-тега Марко. Так как текст должен добавляться при форматировании сообщения, мы снова возвращаемся к коду действия `postItButtonPressed..`

```
- (IBAction)postItButtonPressed:(id)sender {
    NSLog(@"Post It button was pressed: %@", self.tweetTextView.text);

    NSString _____ = [_____ stringWithFormat:@"_____",
    _____];

    SLComposeViewController *composer = [SLComposeViewController compos
eViewControllerForServiceType:SLServiceTypeTwitter];
    [composer setInitialText:tweetText];
    [self presentViewController:composer animated:YES completion:nil];
}
```





Магниты для iOS. Решение

Нужно включить код добавления хеш-тега Марко и проверить, не превышает ли длина сообщения 135 символов (для Twitter; у Facebook ограничений нет).

```
- (IBAction)postItButtonPressed:(id)sender {
    NSLog(@"Post It button was pressed: %@", self.tweetTextView.text);
    NSString *tweetText = [NSString stringWithFormat:@"%@ #MarcoPollo",
self.tweetTextView.text];

    SLComposeViewController *composer = [SLComposeViewController compos
eViewControllerForServiceType:SLServiceTypeTwitter];
    [composer setInitialText:tweetText];
    [self presentViewController:composer animated:YES completion:nil];
}
```

Не пропустите
эту запятую!

Сначала нужно построить текст, кото-
рый будет использоваться в сообщении.
Мы инициализируем объект NSString для
объединения хеш-тега с текстом, вве-
денным пользователем.

animated

.text

setInitialText



ТЕСТ-ДРАЙВ

- 1 Введите код из упражнения с магнитами.
- 2 Постройте и запустите приложение.
Продолжайте, добавьте еще один твит! И замечательную подпись #MarcoPollo в конце.
- 3 Отведайте курицы!





Ваш инструментарий разработки для iOS

Глава 2 осталась позади, а ваш инструментарий пополнился некоторыми базовыми паттернами и синтаксическими конструкциями.

Раскадровка

- ☐ Содержит описание всех представлений вашего приложения.
- ☐ Вся работа по определению макетов представлений ведется в редакторе раскадровки Storyboard Editor.
- ☐ Xcode поддерживает возможность относительного позиционирования при разработке приложений iOS в редакторе раскадровки.

Элементы управления

- ☐ Предназначены для взаимодействия с представлением.
- ☐ Находятся под управлением контроллеров, которые представляют собой классы.
- ☐ Эти классы разделены на файлы .h (заголовок) и .m (реализация).

КЛЮЧЕВЫЕ МОМЕНТЫ



- Действия реагируют на события.
- Ссылки IBOutlet используются для работы с пользовательским интерфейсом для вывода или получения информации от пользователя.
- Редактор раскадровки Storyboard Editor автоматически генерирует

большую часть кода, необходимого для создания действий и ссылок IBOutlet.

- Объявления свойств используются для создания как переменных экземпляров, так и методов чтения и записи этих переменных.

2.5 интермедия

Синтаксис

Понимаю, стенография и все такое, но ведь ни единого слова не разберешь...



Пора заняться мелочами. Мы написали пару приложений и более или менее разобрались с общей картиной. Пришло время досконально, построчно разбираться в мелочах. Почему в коде повсюду разбросаны символы @? Чем метод отличается от сообщения? Как именно работают свойства? Нас ждет краткая экскурсия по синтаксису Objective-C; потом можно будет вернуться к построению приложений.

Классы: интерфейс и реализация

В Objective-C определение класса состоит из двух файлов, *.h* и *.m*. Файл *.h* определяет, как другие объекты должны взаимодействовать с вашим классом. Он определяет свойства, методы и другие атрибуты.

Файл *.m*, импортирующий файл *.h*, реализует внутреннее поведение класса. Компилятор проверяет, что аспекты класса, определяемые в заголовочном файле интерфейса, имеют реализацию в *.m*.

Здесь начинается объявление интерфейса класса...

```
#import <UIKit/UIKit.h>

Имя нового класса
@interface ViewController : UIViewController
@property (weak, nonatomic) IBOutlet UITextView *tweetTextView;

- (IBAction)postItButtonPressed:(id) sender;

@end
```

Класс, производным от которого является новый класс

Методы и свойства располагаются между директивами @interface и @end.

...а здесь оно заканчивается.



ViewController.h

В приложении Марко здесь ничего не используется, поэтому этот фрагмент шаблонного кода остается пустым.

```
#import "ViewController.h"
#import "Social/Social.h"

@interface ViewController ()

@end

@implementation ViewController
...
@end
```

Вы можете добавить расширение класса для объявления логики, которая будет использоваться только как часть файла реализации.

Здесь размещается реализация методов.

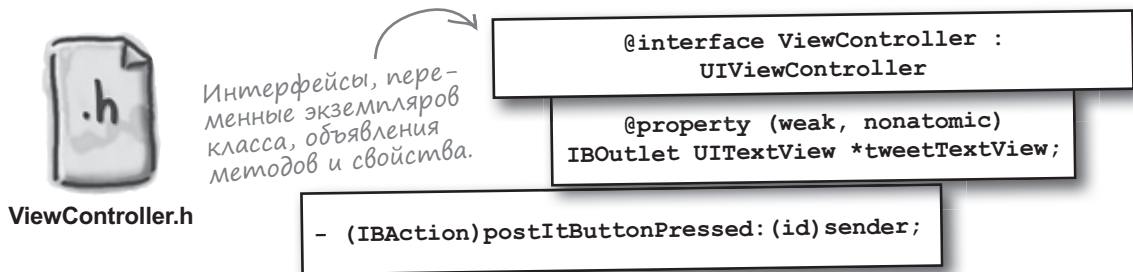


ViewController.m

Заголовочные файлы описывают интерфейс класса

В Objective-C при определении класса их интерфейс выделяется в заголовочный файл. Здесь указывается, создается ли данный класс посредством наследования от других классов, а также перечисляются переменные экземпляров, свойства и методы класса.

С развитием Objective-C часть кода стала перемещаться в файл реализации как расширение класса. Это единственный способ определения действительно «закрытых» свойств и методов, ограниченных от постороннего доступа.



Возьми в руку карандаш



Файл *ViewController.h* из приложения Марко. Заполните пустые места и объясните, что делает каждая строка.

```
..... #import <UIKit/UIKit.h>

.....

..... @interface ViewController : UIViewController
..... @property (weak, nonatomic) IBOutlet
..... UITextView *tweetTextView;

.....

..... - (IBAction)postItButtonPressed: (id) sender;

.....

..... @end
```



ViewController.h

Возьми в руку карандаш

Решение

```
#import <UIKit/UIKit.h>
```

Эта строка почти идентична директиве `#include` в языке C — не считая того, что она автоматически предотвращает многократное включение одного заголовка (так что проверки `#ifndef MY_HEADER` становятся лишними).

Директива `# import` встраивает другой файл (почти всегда заголовочный) в текущий файл во время компиляции. Она используется для включения классов, констант и т. д. и других файлов.

`@interface` означает, что мы собираемся объявить класс.

Затем указывается имя класса, и если он наследует от другого класса — двоеточие и имя супер-класса.

Objective-C не поддерживает множественное наследование...

```
@interface ViewController : UIViewController {
```

Здесь указывается, от какого класса наследует текущий класс и с какими протоколами он совместим.

```
UITextField *notesField_;
```

Синтаксис переменных экземпляров не отличается от C++: базовые типы (такие, как `int` и `float`) используются без изменений, а указатели помечаются звездочкой. По умолчанию всем полям назначается защищенный уровень доступа, но его можно изменить при помощи ключевых слов `@private` и `@public`, как и в C++.

Здесь объявляются переменные экземпляров класса; в приложении Марко их не было, но вы понимаете, что имеется в виду.

```
}
```

После закрытия секции полей в интерфейсе следуют объявления свойств. Директива `@property` сообщает Objective-C о том, что для свойства следует сгенерировать методы доступа, для вызова которых будет использоваться «точечная» запись.



ViewController.h

Возьми в руку карандаш

Решение

Ключевое свойство `@property` сообщает компилятору о том, что для этого свойства будет создан метод чтения и (возможно) метод записи.

Атрибуты свойств; мы поговорим о них более подробно...

```
@property (weak, nonatomic) IBOutlet UITextView *tweetTextView;
```

Interface Builder по маркеру `IBOutlet` опознает свойства, связываемые с элементами управления (как в случае со свойством `notes` в `InstaTwit`).

Тип и имя свойства (по аналогии с полями класса).

Знак «минус» означает, что метод является методом экземпляра (тогда как знаком «плюс» обозначается метод класса). Все методы в Objective-C являются открытыми.

Объявления методов.

```
- (IBAction)postItButtonPressed: (id) sender;
```

Маркер `IBAction` используется для пометки событий.

Сигнатура методов `IBAction` может не содержать аргументов, иметь один аргумент типа `ID` (аналог ссылки на `Object` в языке `Java`) или два аргумента, в одном из которых передается идентификатор отправителя, а в другом — `UIEvent*` для события, инициировавшего вызов.

@end

@end: завершает объявление интерфейса класса.



ViewController.h

Свойства используются для эффективности

Свойства позволяют создать методы чтения и записи для работы с объектом; они также предоставляют дополнительную помощь. Так как создание этих методов поручается компилятору, последний также может убедиться в том, что все свойства используются в коде, и синтезировать методы доступа. И все это в одной строке кода!

```
@property (weak, nonatomic) IBOutlet UITextView *tweetTextView;
```

Атрибуты свойства



ViewController.h

Атрибуты свойств передают информацию компилятору

Чтобы указать, как именно должны работать методы доступа, вы объявляете атрибуты свойств. Атрибуты делятся на три категории: возможность записи, семантика присваивания и атомарность. По умолчанию свойства атомарны. Это означает, что синтезированные методы доступа всегда возвращают или присваивают значение переменной способом, безопасным в условиях многопоточного выполнения.

Полный список атрибутов можно найти в документации разработчика Apple.



Свойства безопасны,
потому что мы управляем
доступом к нашим пере-
менным, верно?

**Верно! Свойства защищают
переменные.**

Определяя свойство, мы ограничиваем возможность взаимодействия объектов с данными нашего класса. Это способствует улучшению инкапсуляции и повышению контроля за работой с данными.

★ КТО И ЧТО ДЕЛАЕТ? ★

Ниже перечислены наиболее часто используемые атрибуты свойств и их определения. Соедините каждый атрибут с его определением.

read-only

Вы хотите, чтобы свойство могло изменяться. Компилятор генерирует за вас методы чтения и записи. Атрибут используется по умолчанию.

strong

Атрибут не продлевает жизнь объекта, на который указывает ссылка. При отсутствии сильных ссылок на объект ему присваивается nil. Сторона, получающая такую ссылку на объект, не становится его владельцем.

read/write

Используется при работе со значениями объекта. Объект продолжает «жить» до тех пор, пока в коде существует хотя бы одна сильная ссылка на него. Создавая такую ссылку, вы становитесь владельцем объекта.

copy

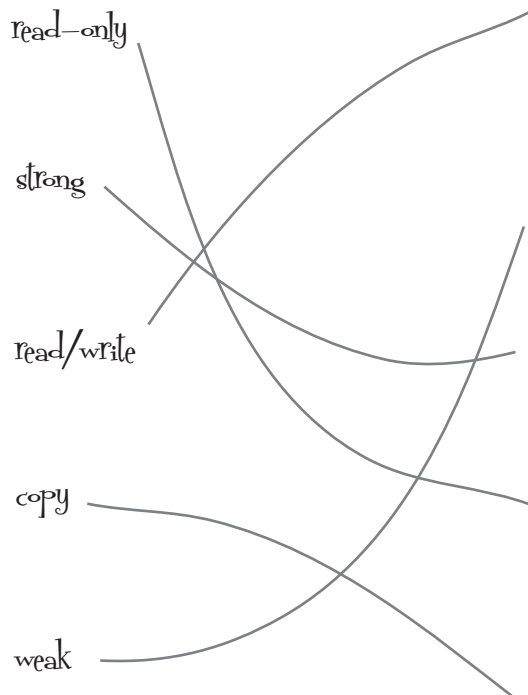
Вы не хотите, чтобы свойство могло изменяться извне. Значение поля, для которого создано свойство, все еще может изменяться, но компилятор не генерирует метод записи.

weak

В свойстве должна храниться копия значения вместо самого значения; например, если содержимое исходного массива не должно изменяться после присваивания. Передаваемому значению отправляется сообщение копирования, результат которого и удерживается в дальнейшем.

КТО И ЧТО ДЕЛАЕТ? РЕШЕНИЕ

Ниже перечислены наиболее часто используемые атрибуты свойств и их определения. Соедините каждый атрибут с его определением.



Вы хотите, чтобы свойство могло изменяться. Компилятор генерирует за вас методы чтения и записи. Атрибут используется по умолчанию.

Атрибут не продлевает жизнь объекта, на который указывает ссылка. При отсутствии сильных ссылок на объект ему присваивается nil. Сторона, получающая такую ссылку на объект, не становится его владельцем.

Используется при работе со значениями объекта. Объект продолжает «жить» до тех пор, пока в коде существует хотя бы одна сильная ссылка на него. Создавая такую ссылку, вы становитесь владельцем объекта.

Вы не хотите, чтобы свойство могло изменяться извне. Значение поля, для которого создано свойство, все еще может изменяться, но компилятор не генерирует метод записи.

В свойстве должна храниться копия значения вместо самого значения; например, если содержимое исходного массива не должно изменяться после присваивания. Передаваемому значению отправляется сообщение копирования, результат которого и удерживается в дальнейшем.

Часто задаваемые вопросы

В: Как компилятор узнает, какое поле будет использоваться для хранения значения?

О: Когда компилятор автоматически синтезирует свойство, он генерирует для него переменную экземпляра. Имя переменной состоит из символа подчеркивания и имени свойства. Например, для свойства с именем `superSecretField` будет создана базовая переменная с именем `_superSecretField`. Имя переменной можно изменить, добавляя директиву `@synthesize`, но мы рекомендуем использовать имя по умолчанию.

В: Как насчет ключевого слова `nonatomic`?

О: По умолчанию генерируемые методы доступа обладают потоковой безопасностью и используют мьютексы при изменении значения свойства. Такой метод доступа называется **атомарным (atomic)**. Если ваш класс не будет использоваться в многопоточных условиях, безопасность оборачивается тратой ресурсов. Вы можете приказать компилятору исключить проверку мьютексов, объявляя свойство с атрибутом `nonatomic`. Учтите, что простое объявление атомарных свойств не обеспечивает потоковой безопасности всего класса, так что будьте внимательны.

Передача сообщений: как это происходит в Objective-C

Чтобы взаимодействовать с другим объектом в Objective-C, вы передаете ему сообщение. Сообщение может содержать аргументы (а может и не содержать), а совокупность имени метода и аргументов называется селектором. Помните: сообщения отправляются другому объекту, где они обрабатываются реализацией метода.

Ниже приведено сообщение, используемое с элементом управления (мы использовали его в InstaTwit; он выглядит как вращающееся колесо). Метод возвращает количество строк в заданном компоненте. Его объявление выглядит так:

Возвращаемый тип Имя метода Тип первого аргумента Локальное имя аргумента Открытое имя второго аргумента

```
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger) component;
```

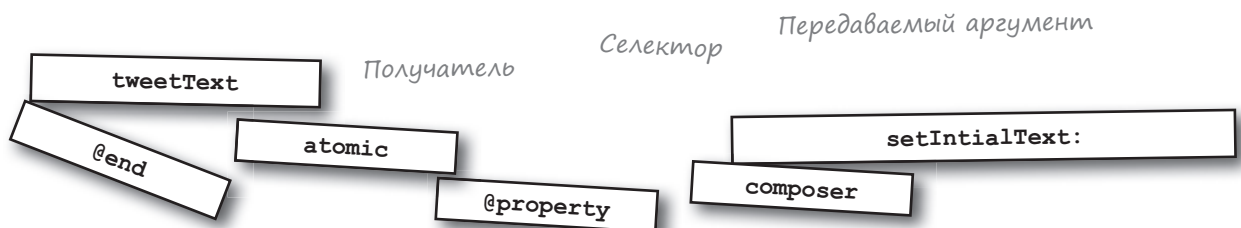
Тип второго аргумента Локальное имя второго аргумента



Развлечения с магнитами

Перед вами сообщение из файла *ViewController.m* приложения Марко; оно приказывает представлению composer задать его исходный текст. Постройте само сообщение и добавьте описания его фрагментов.

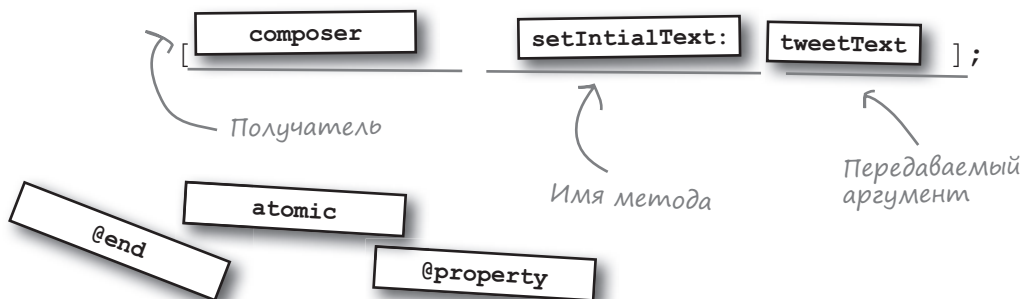
[_____];





Развлечения с магнитами. Ответ

Перед вами сообщение из файла *ViewController.m* приложения Марко; оно приказывает представлению composer задать его исходный текст. Постройте само сообщение и добавьте описания его фрагментов.



Часть Задаваемые Вопросы

В: Что происходит с аргументами методов? Зачем там два имени, перед двоеточием и после типа?

О: В Objective-C для аргументов можно указывать два имени, открытое и локальное. Открытое имя становится частью селектора, когда кто-либо хочет отправить сообщение вашему объекту. Это имя указывается перед двоеточием. Имя после типа относится к локальной переменной, в которой хранится значение. В Objective-C эти имена не обязаны совпадать, так что вы можете использовать содержательное открытое имя для людей, которые будут использовать ваш класс, и удобное локальное имя для своего кода.

В: Вы упомянули о селекторах, но я все еще плохо понимаю, что это такое.

О: Селекторы — уникальные имена методов, когда Objective-C преобразует сообщение в фактический вызов метода. Например, найдите код, в котором используется селектор `pickerView:numberOfRowsInComponent`. Мы еще вернемся к селекторам, когда будем заниматься связыванием элементов интерфейса с программным кодом. А пока Interface Builder делает все за нас.

В: При отправке сообщения `resignFirstResponder` используется тип отправителя «id». Что это значит?

О: «id» — тип Objective-C, который может представлять любой объект Objective-C. Можно сказать, что это аналог `void*` в языке C++. Так как Objective-C является языком с динамической типизацией, отправка сообщений объекту типа «id» вполне допустима. Исполнительная среда определит, способен ли объект отреагировать на отправленное сообщение.

В: А что произойдет, если объект не может отреагировать на сообщение?

О: Произойдет исключение. Собственно, именно по этой причине рекомендуется по возможности использовать переменные с сильной типизацией — это повышает вероятность обнаружения ошибок по предупреждениям во время компиляции, а не только по исключениям во время выполнения. Однако в некоторых ситуациях обобщенные типы очень удобны — например, для реализации методов обратного вызова, когда отправителем может быть любой из множества объектов.

В: Все так серьезно — квадратные скобки для передачи сообщений?

О: Да. И для индексирования массивов. Привыкайте, это неизбежно.

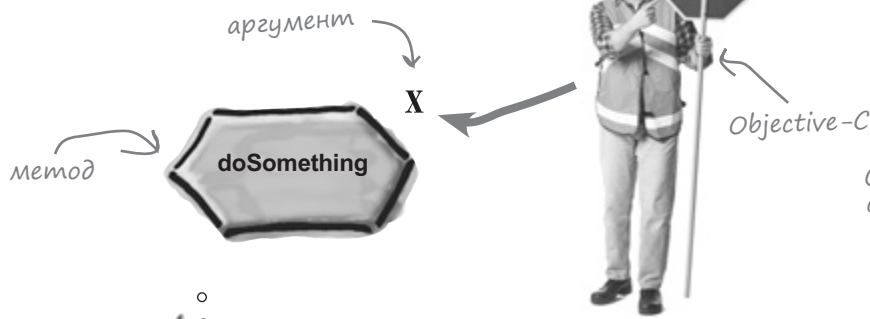
Вы говорите о каких-то «сообщениях», а по-моему, это самый обычный вызов метода. Вам не кажется, что это глупо?



Сообщение — запрос на вызов метода с конкретным именем и с конкретными аргументами. В Objective-C приложение отправляет сообщения объектам, а те реагируют на полученные сообщения.

Исполнительная среда Objective-C преобразует сообщение в вызов метода, который возвращает значение. Таким образом, обычно мы говорим об отправке некоторого сообщения, но когда вы реализуете то, что происходит при получении сообщения, вы реализуете метод.

```
[foo: dosomething:x];
```



Objective-C
пытается связать
сообщение
с существующим
методом... И это не
всегда получается!

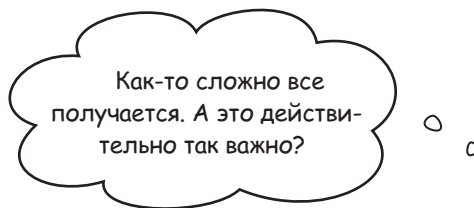
Objective-C создает селектор на основе вашего сообщения и пытается подобрать соответствующий метод.



РАССЛАБЬТЕСЬ

Возможно, что-то из сказанного вам покажется непонятным. Тема достаточно глубокая... Но дальше будет проще!

Сейчас мы блуждаем в дебрях, пытаясь как-то разобраться во всех этих точках с запятой и обрывках текста. Пора взяться за очередное приложение, чтобы подкрепить теорию практикой...



Сообщения позволяют определять получателей во время выполнения, а не во время компиляции.

Отправка сообщения — это механизм организации взаимодействия объектов друг с другом. С их помощью вы приказываете объекту выполнить операцию или вычислить некоторое значение. Даже обращение к свойству преобразуется компилятором в сообщение.



Документация Apple основана на терминологии с сообщениями.

КЛЮЧЕВЫЕ МОМЕНТЫ



- В Objective-C приложение отправляет получателям сообщения. Исполнительная среда преобразует их в вызовы методов.
- Объявления методов размещаются в заголовочном (.h) файле после закрывающей фигурной скобки интерфейса.
- Реализации методов размещаются в файле реализации (.m) между директивами @implementation и @end.
- Для аргументов методов обычно определяются имена, которые используются при отправке сообщений.
- Аргументы могут иметь два имени: внешнее и внутреннее.
- Методы экземпляров помечаются маркером "-"; статические методы помечаются маркером "+".

Раз уж мы занялись сообщениями....



Пора снова
поработать...



Ваш инструментарий синтаксиса

Глава 2.5 осталась позади, а ваш инструментарий разработки дополнился некоторыми синтаксическими конструкциями.

Синтаксис сообщения:

`[composer setInitialText:tweetText];`

Получатель Метод Передаваемый аргумент

Классы делятся на:

- ☐ Интерфейсы (которые обычно находятся в заголовочном файле (.h), но могут находиться в файле .m как расширение класса)
- ☐ Реализации (всегда находятся в файле .m)

Свойства:

- ☐ Обладают атрибутами.
- ☐ Атрибуты используются для генерирования методов доступа (чтения и записи).
- ☐ Способствуют инкапсуляции функциональности.

КЛЮЧЕВЫЕ МОМЕНТЫ



- В Objective-C приложение отправляет получателем сообщения. Исполнительная среда преобразует их в вызовы методов.
- Объявления методов размещаются в заголовочном (.h) файле после закрывающей фигурной скобки интерфейса.
- Реализации методов размещаются в файле реализации (.m) между директивами @implementation и @end.
- Для аргументов методов обычно определяются имена, которые используются при отправке сообщений.
- Аргументы могут иметь два имени: внешнее и внутреннее.
- Методы экземпляров помечаются маркером "-"; статические методы помечаются маркером "+".

3 Таблицы, представления и данные

Таблица и представление

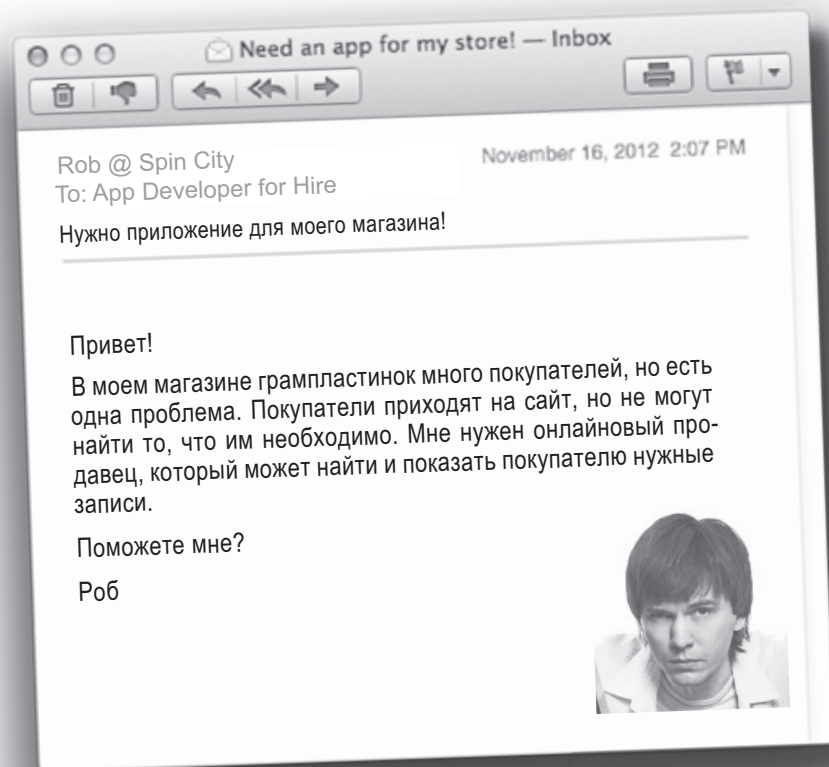
Мне кофе с двумя кусочками сахара, сливками, немного корицы, дважды перемешать, потом...



В большинстве приложений iOS используется несколько представлений. Мы написали симпатичное приложение с одним представлением. И все же каждый, кто когда-либо пользовался смартфоном, знает, что типичное представление одним представлением не ограничивается. Самые впечатляющие приложения iOS используют при работе со сложной информацией несколько представлений. Мы начнем с навигационных контроллеров и табличных представлений наподобие тех, которые используются в приложениях Mail и Контакты. Вот только у нас они будут использоваться несколько иначе...

Поздравляем!

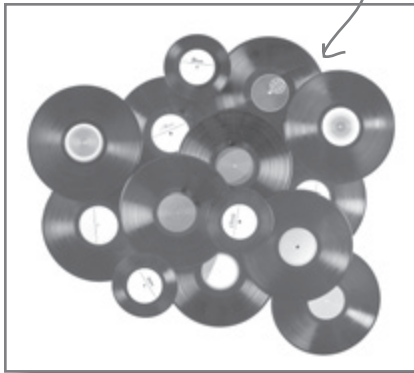
Вам поручили новую работу — создание приложения для SpinCity, модного магазина грампластинок. Владелец магазина, Роб, прислушивается к мнению своих покупателей и обращается к вам с просьбой...



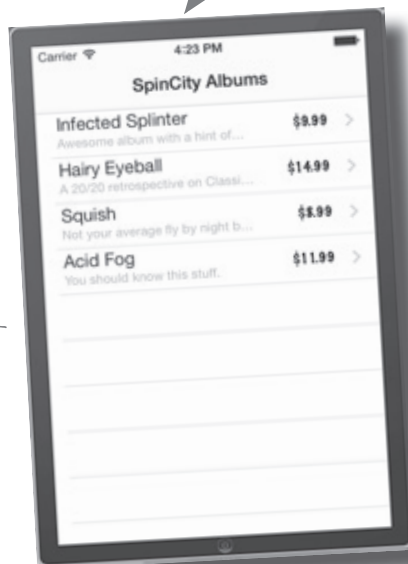
Приложение: каталог SpinCity

Приложение должно хранить данные о пластинках, имеющихся в магазине, выводить их для покупателей с возможностью поиска. Так Робу будет проще разобраться в его виниловых сокровищах, а поток посетителей начнет постепенно превращаться в денежный поток.

Каждая пластинка хранится на определенном месте в магазине.



Продавец вводит все эти данные.



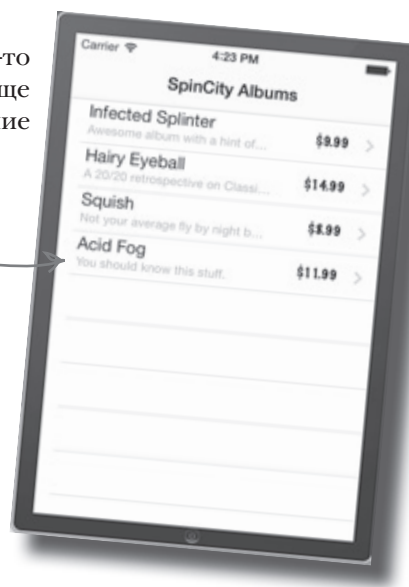
Hi-Fi-приложение адресовано счастливым покупателям!

Вам поручено создать приложение для iPhone, которое будет выводить информацию о каждой пластинке: местонахождение, цену и описание. Приложению понадобится полный список всех пластинок.

Как работают приложения iOS

У нас довольно много информации, которую нужно как-то передать пользователю. С местонахождением и ценой еще можно управиться, но сколько-нибудь подробное описание в строке таблицы точно не поместится.

Каждая пластинка представляется ячейкой, но на экране не хватит места для всей необходимой информации! Придется создать дополнительное представление для подробного описания.



Сенсорный экран

Приложения iOS стали одной из первых реализаций пользовательского интерфейса, основанного на прикосновениях к сенсорному экрану. По этой причине, а также из-за малого размера доступного экрана для одной задачи приложения с несколькими представлениями стали обычным явлением. Приложение SpinCity должно реагировать на прикосновение пользователя к экрану, выдавая дополнительную информацию.



В приложениях iOS с несколькими представлениями обычно используется одно «главное» (Master) представление и несколько «детализированных» (Detail) представлений, предназначенных для вывода расширенной информации. В нашем приложении табличное представление выполняет функции главного; также понадобится детализированное представление для вывода всей информации, которую нужно донести до пользователя.



Развлечения с магнитами

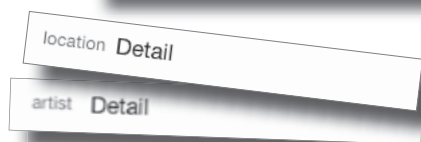
Мы уже показали, как будет выглядеть первое представление. А из каких частей вы бы собрали второе, детализированное представление?



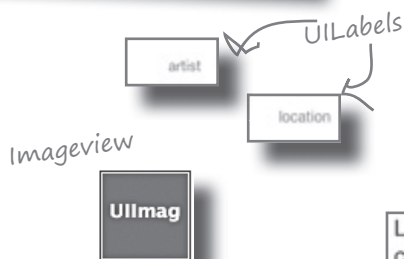
Клавиатура



Представление #2



Панель навигации



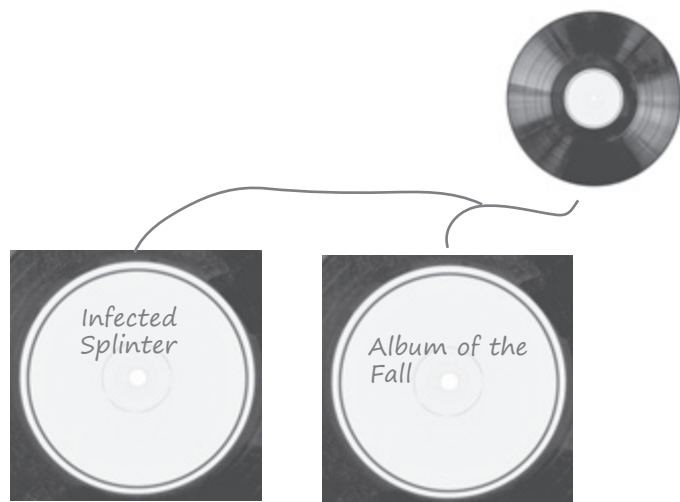
UITextView
с текстом-
заполнителем

Lorem ipsum dolor sit er elit lamet,
consectetur cillum adipisicing pecu, sed
do eiusmod tempor incididunt ut labore et

Если вы
затрудняетесь
с ответом,
переверните
страницу — там
есть подсказка!

Иерархические данные — за пределами табличного представления

Данные часто образуют отношения «родитель—потомок»: высокоуровневый блок данных с дополнительными уточняющими данными. Например, эта модель часто встречается в приложениях для работы с контактной информацией и календарем. В нашем случае информация о пластинках хорошо вписывается в иерархическую модель.



Данные верхнего уровня — название пластинки.

Детализированные данные состоят из информации на наклейке, названий песен, описания и местонахождения пластинки.

Используйте сочетание табличного и детализированных представлений для представления иерархических данных

Сочетание табличного представления с детализированным идеально подходит для таких видов данных. В табличном представлении выводятся все элементы верхнего уровня (в нашем случае названия пластинок), а в детализированном представлении выводится подробная информация о каждой пластинке.

При построении приложений следует тщательно продумать, как будет использоваться каждое представление: какую информацию вы хотите сообщить пользователю? Как пользователь будет работать с этой информацией? Разные представления часто ориентируются на разные сценарии использования, поэтому, возможно, вы захотите добавить дополнительные кнопки (или, наоборот, заблокировать часть функций) в зависимости от того, в каком представлении находится пользователь и что он пытается сделать.





Откуда я это узнаю? Может, существуют какие-то шаблоны для таких приложений? И разве программист не сам выбирает, как должно себя вести его приложение?

Компания Apple достаточно подробно описывает поведение приложений iPhone в документации HIG.

iOS Human Interface Guidelines (HIG) — поддерживаемый компанией Apple документ с перечнем рекомендуемых правил и стандартных решений, относящихся к пользовательскому интерфейсу, для приложений App Store.

Когда вы приступите к вступительной части, обратите внимание на примечание об использовании навигации в иерархических данных (с детализированным представлением). Так как эта схема взаимодействия встречается очень часто, в среде Xcode для нее предусмотрен специальный шаблон приложения Master-Detail.

Документацию HIG можно найти в Интернете по адресу <https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/> или в окне Organizer среды Xcode (выполните поиск по строке «iOS Human Interface Guidelines»).

Часть Задаваемые Вопросы

В: Соблюдать правила HIG обязательно?

О: И да, и нет. Всегда полезно следить за тем, чтобы в вашем приложении соблюдались «правила iOS»; вы упростите жизнь будущим пользователям и рецензентов Apple, которые будут проверять ваше приложение при отправке в App Store.

Конечно, в приложении есть место для оригинальности, но для базовых операций (таких, как перемещение по иерархическим данным) лучше придерживаться единых правил, чтобы приложение соответствовало ожиданиям пользователя. А творческие наклонности проявить в областях, специфических для вашего приложения.

В: Зачем переключать представления? Почему бы не изменить текущее представление для вывода дополнительной информации?

О: Это одна из областей, в которых важную роль играют малый размер экрана и схемы взаимодействия, принятые в iOS. Из-за ограниченности экранного пространства приложения iOS сильно ориентированы на решение конкретных задач. Одна из самых сложных проблем построения хорошего приложения — эффективная передача необходимой информации без «информационной перегрузки» или чрезмерного усложнения взаимодействий. Каждое представление должно решать конкретную задачу, а при его проектировании необходимо подумать о том, как сообщить пользователю нужную информацию простым и понятным способом.

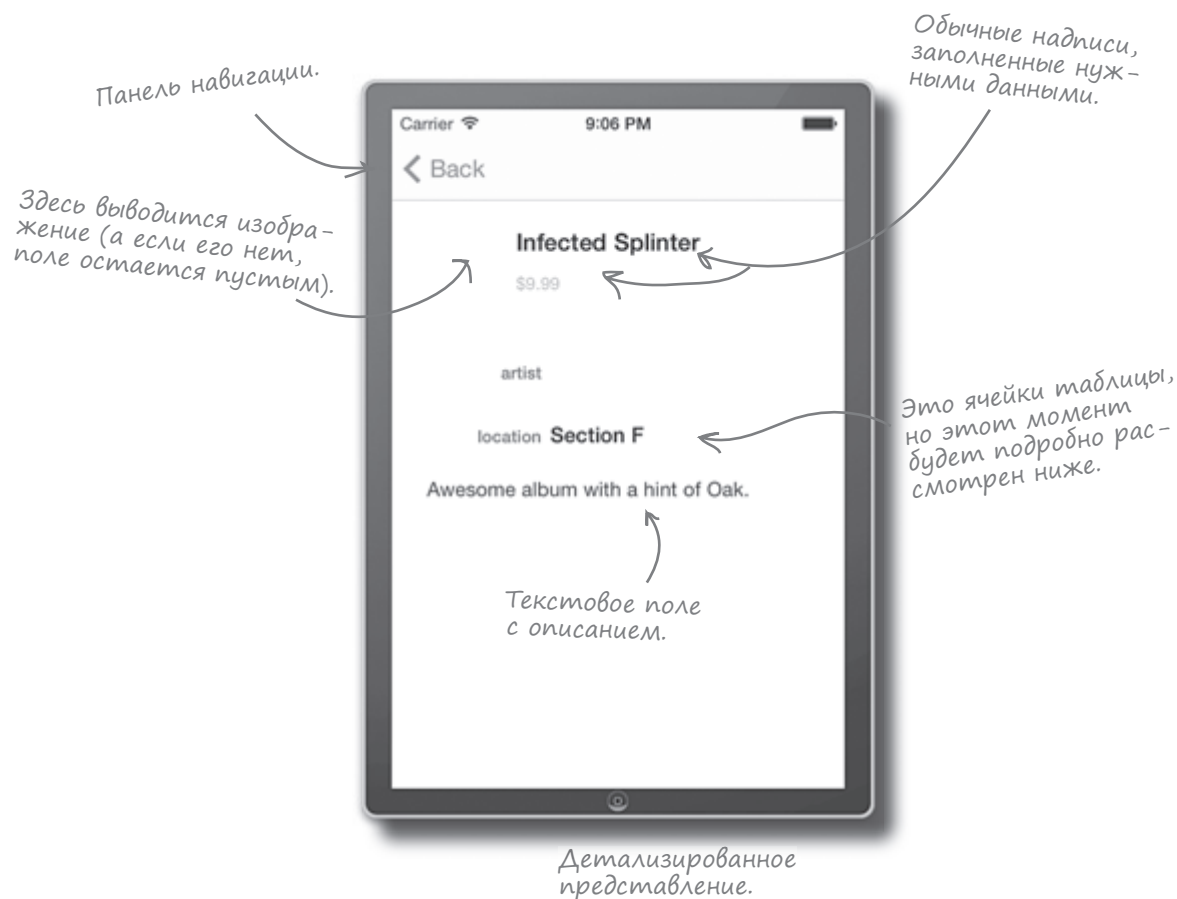
В: Как еще сенсорный экран влияет на интерфейс приложений?

О: При проектировании для сенсорного экрана некоторые аспекты приложения полностью изменяются (скажем, по сравнению с веб-программированием); важнейшее отличие заключается в том, что пространство взаимодействия определяется размерами кончика пальца, а не указателя мыши. Также следует учитывать и другие аспекты — например, в интерфейсе перестает работать такая привычная парадигма дизайнера, как наведение (hover).



Развлечения с магнитами — решение

Вот какое детализированное представление получилось у нас. О том, как реализовать его, мы расскажем чуть позже.



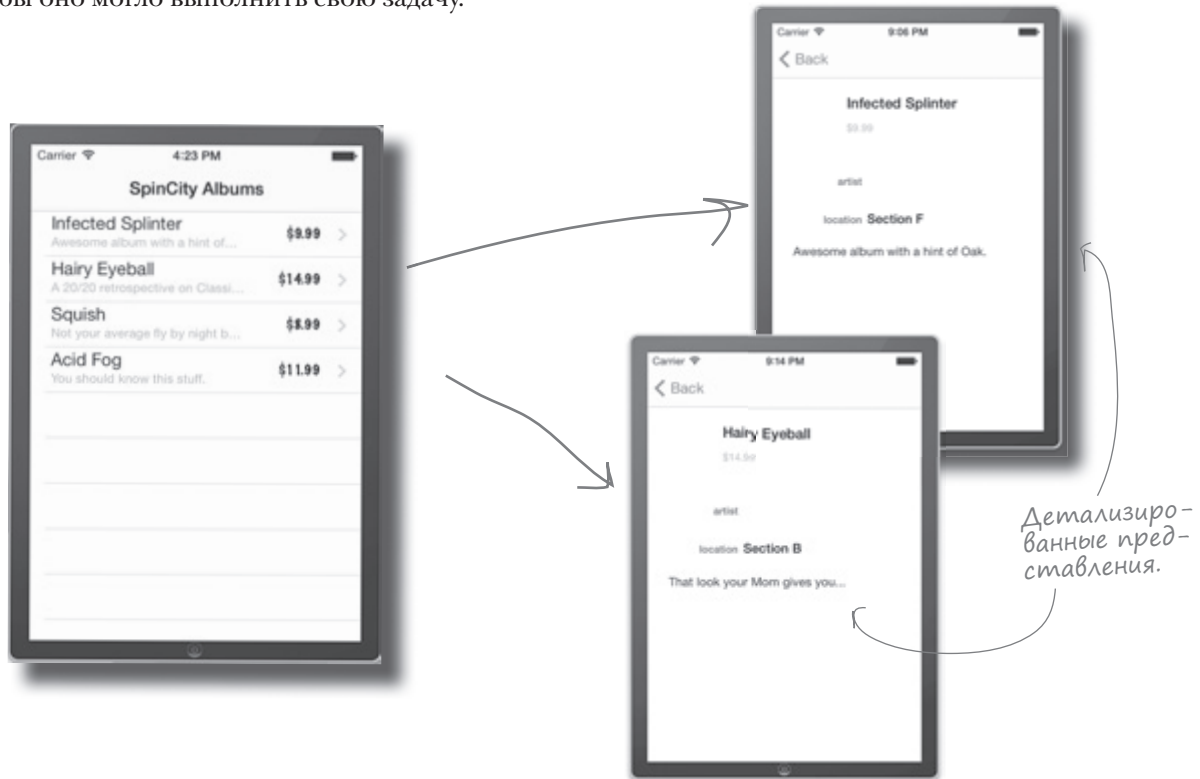
РАССЛАБЬТЕСЬ

Такое детализированное представление получилось у нас. Если вы предложили нечто иное — это нормально.

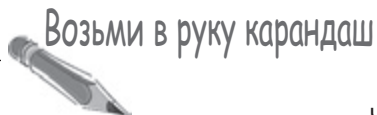
Мы взяли за образец стандартное приложение Контакты, но оригинальность приложения определяется его стилем и особенностями. Будьте осторожны, определяя соотношение между оформлением и функциональностью, но обязательно придайте своему приложению толику оригинальности. О том, как реализовать все это, мы расскажем далее.

А теперь нужно связать эти два представления...

У нас есть два представления, но они никак не связаны между собой. Мы должны указать, когда должно использоваться каждое из этих представлений, как должен происходить переход между ними и какая информация необходима каждому из представлений для того, чтобы оно могло выполнить свою задачу.

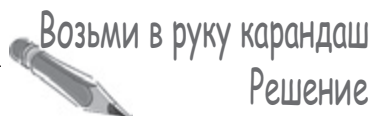


Как таблица узнает, какие данные показывать в детализированном представлении? И как пользователь возвращается к главному представлению?



Что нужно подключить, чтобы схема с двумя представлениями заработала?

- | | |
|--|---|
| <input type="checkbox"/> Панель вкладок в нижней части главного представления. | <input type="checkbox"/> Кнопки навигации. |
| <input type="checkbox"/> Вспомогательное приложение. | <input type="checkbox"/> Обработка прикосновений к ячейкам таблицы. |



Решение

Что нужно подключить, чтобы схема с двумя представлениями заработала?

- ☐ Панель вкладок в нижней части представления.

Панель вкладок используется для управления несколькими равноправными представлениями, которые могут быть не связаны друг с другом. Полезно, но не в нашем случае!

- ☐ Вспомогательное приложение.

Термином «вспомогательное приложение» обычно описывается узкоспециализированное приложение, поддерживающее только одно или два представления — как, например, стандартное приложение Погода или Акции для iOS.

- ☒ Кнопки навигации.

Кнопки возврата и перехода между представлениями становятся частью контроллера навигации

- ☒ Обработка прикосновений к ячейкам таблицы.

Прикосновение пользователя к ячейке таблицы инициирует событие, которое сообщает приложению о случившемся. Мы можем воспользоваться этим событием для связывания представлений.

Контроллер навигации обеспечивает перемещение между представлениями

Контроллер навигации может управлять переходами между представлениями. Встроенные переходы уже существуют, нужно лишь связать компоненты. Контроллер предоставляет такие функции, как кнопки возврата, кнопки заголовка и историю просмотра, чтобы пользователь мог перемещаться по данным, не рискуя заблудиться. Контроллер навигации берет на себя такие аспекты, как определение размеров представлений и ведение стека представлений; вам остается лишь сообщить ему, с какими представлениями вы намерены работать.



Класс UINavigationController поддерживает идею жественных представлений, может переключаться между ними с отображением анимаций, а также имеет встроенную поддержку кнопок на панели навигации.

Часть Задаваемые Вопросы

В: Какие стандартные приложения iOS используют контроллер навигации?

О: Чтобы получить представление о том, как работает контроллер навигации, посмотрите к приложению Почта на iPhone и iPad. На iPad используется контроллер навигации, сходный с iPhone, но позволяющий одновременно отображать как главное, так и детализированное представление. Это позволяет iPad использовать дополнительное экранное пространство без изменения схемы навигации между устройствами.

Другой хороший пример — приложение Календарь. Хотя оно поддерживает несколько представлений (месяц, день, список или неделя), по сути все равно управляется контроллером приложений.

В: Обязательно ли использовать табличное представление в качестве корневого?

О: Нет, это всего лишь самый распространенный вариант, поскольку он предоставляет естественный способ отображения сводки данных, по элементам которой пользователь может запросить дополнительную информацию. Кроме того, табличные представления легко настраиваются, так что в некоторых приложениях их даже не узнать, как, например, в приложении Блокнот или в магазине iTunes.

В: Как контроллер навигации связан с контроллерами представлений?

О: Вскоре эта тема будет рассмотрена куда более подробно, а пока достаточно сказать, что контроллер навигации координирует переходы между контроллерами представлений. Как правило, каждое представление верхнего уровня поддерживается контроллером представления, а с появлением представления на экране соответствующий контроллер представления начинает получать от него события. Существует целый жизненный цикл представлений (мы подробно рассмотрим его), при помощи которого контроллер представления узнает о том, что происходит с его представлением.

В: А если я захочу разместить наверху свои кнопки — как, например, кнопку редактирования в приложении Почта для iPhone? Это не создаст проблемы с контроллерами навигации?

О: Вовсе нет — никаких проблем. Контроллер навигации поддерживает размещение кнопок на верхней панели навигации, нужно лишь приказать ему разместить кнопку в нужном месте. Контроллер также поддерживает кнопку возврата к предыдущему представлению.

В: Вы упомянули о том, что на iPad возможно одновременное отображение главного и детализированного представлений. Это не усложняет программирование?

О: В общем-то нет. На iPad существует элемент управления SplitView, которому передается главное и детализированное представления. Это могут быть те же представления, которые вы используете в своем приложении для iPhone. На iPad эти представления связываются контроллером SplitViewController (вместо контроллера навигации, как на iPhone или iPod Touch). Иногда на iPad можно увидеть главное представление, отображаемое собственным контроллером навигации. Пример такого рода встречается в приложении Почта на iPad. Главное представление (в альбомном режиме) может отображать данные учетных записей с возможностью перехода на уровень папок, а затем на уровень конкретных сообщений. Всем этим уровням соответствуют разные представления в контроллере навигации. Когда вы прикасаетесь к сообщению, в детализированном представлении (большая область справа) отображается полная информация выделенного сообщения.

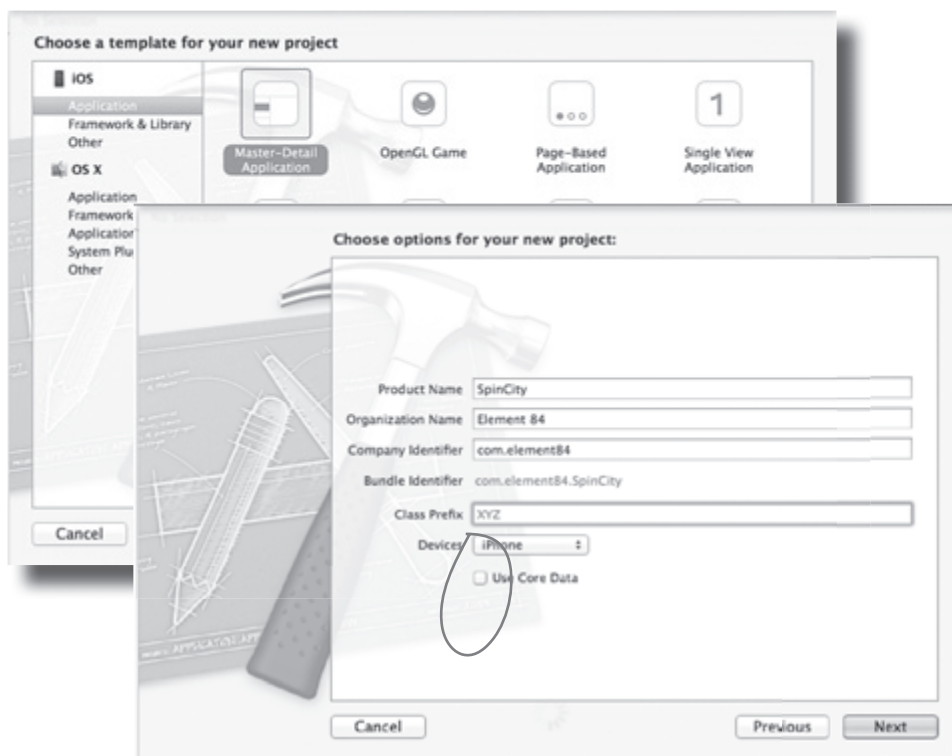
Три представления в одном шаблоне

Итак, вы достаточно хорошо представляете, что будет делать приложение (отображать записи и подробную информацию о них) и как оно будет выполнять свою работу (при помощи контроллера навигации и табличного представления). Теперь можно браться за дело. Запустите Xcode.

1

Выберите шаблон.

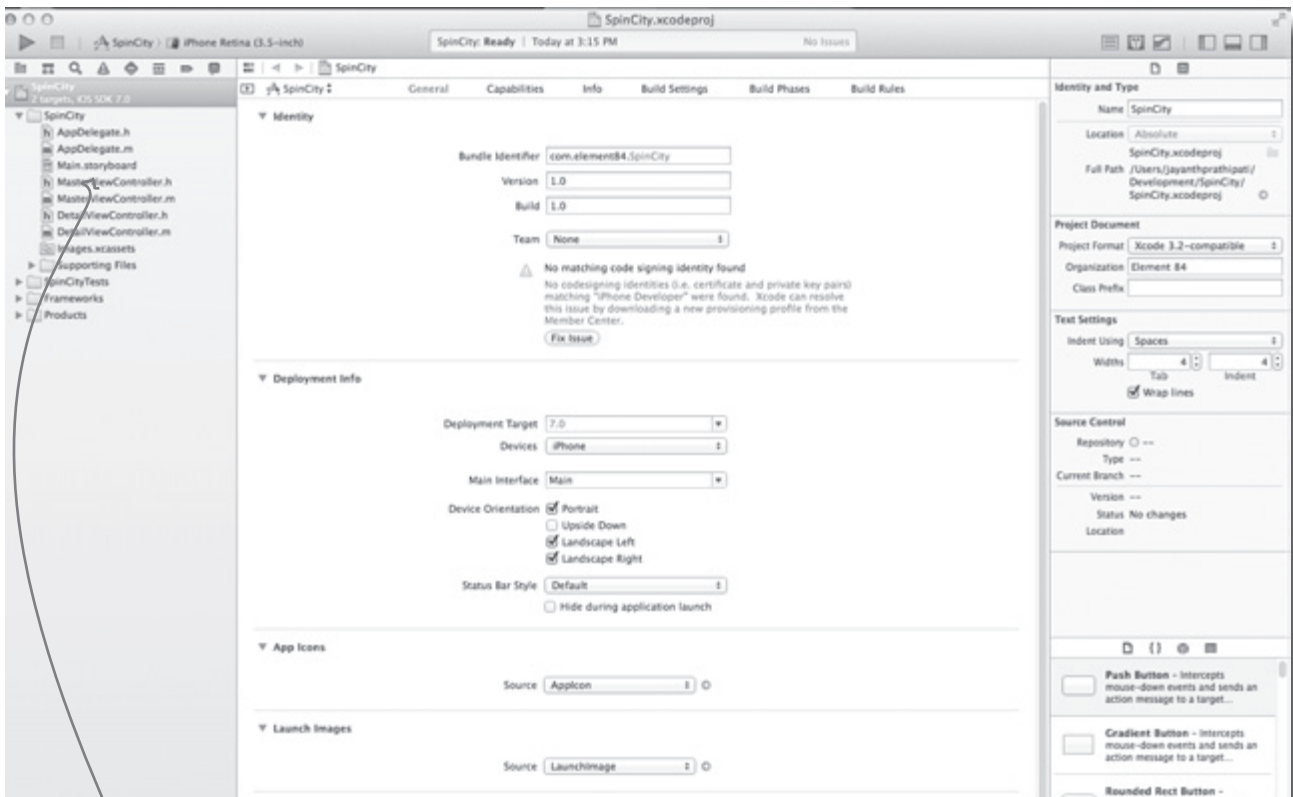
Создайте новый проект и выберите шаблон «Master-Detail Application». В состав этого шаблона входят базовые представления и контроллер навигации. Переименуйте продукт в «SpinCity» без добавления префикса классов. Оставьте флажок «Use Core Data» установленным.



2

Щелкните на кнопке «Next» и сохраните проект.

Сохраните новый проект там, где вам будет удобно. Xcode создает новое приложение iOS с табличным главным представлением и пустым детализированным представлением.



3

Откройте раскладовку приложения.

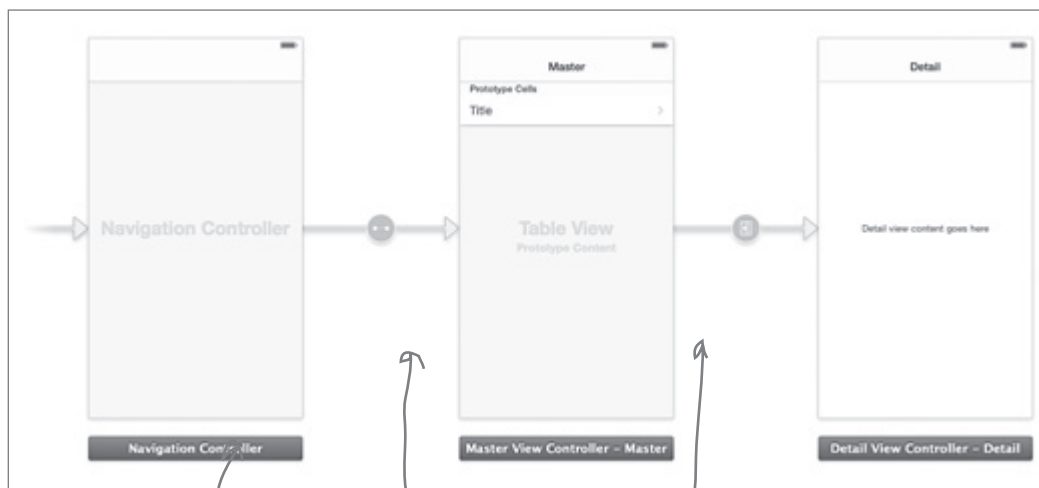
Шаблон включает раскладовку со всеми представлениями проекта, включая переходы. Щелкните на раскладовке; Xcode открывает ее в специальном редакторе. Давайте посмотрим, какие готовые возможности мы получили вместе с шаблоном...



Storyboards под увеличительным стеклом

Раскадровкой (storyboard) называется представление в Xcode, которое появилось в Xcode 4.2 и предназначено для работы с представлениями и переходами приложения на графическом уровне. Сейчас, когда мы переходим к приложению с несколькими представлениями, важно увидеть, как эти приложения связаны между собой.

Каждое представление в приложении называется в раскадровке «сценой» («scene»).



Приложение состоит из трех сцен

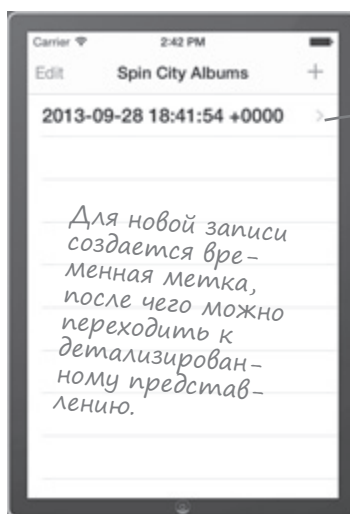
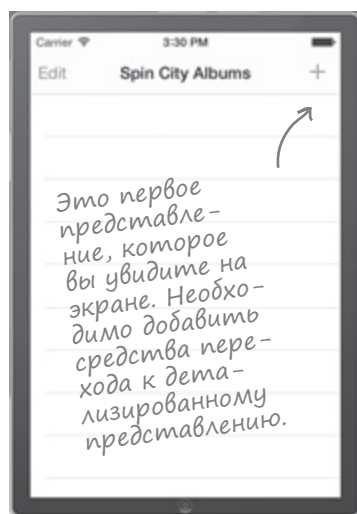
Стрелками обозначаются пути (segues) между представлениями.

Сцены отображают долю контента, предоставляемую контроллером представления. На iPhone это один экран, но на iPad на экране могут одновременно находиться несколько контроллеров (и их представлений). Также на раскадровке изображены **пути**, которые соответствуют различным типам переходов между представлениями. Некоторые пути могут включать модальные, или активные, переходы и могут настраиваться пользователем. Сам путь представляет собой объект, который создается для подготовки нового представления и используется для передачи данных между представлениями.



ТЕСТ-ДРАЙВ

- 1 **Создайте приложение.**
Запустите Xcode и создайте приложение SpinCity (см. инструкции на предыдущей странице). У вас появляется заготовка приложения для дальнейшей работы.
- 2 **Задайте текст заголовка главного представления.**
В редакторе раскадровки щелкните в области навигации контроллера главного представления (Master View Controller). В редакторе свойств (справа) замените значение свойства Title на «SpinCity Albums». Проследите за тем, чтобы флажок «is initial view controller» под заголовком был установлен.
- 3 **Постройте и запустите приложение!**
Чтобы приложение заработало, вам ничего делать не придется; шаблон уже готов к работе. Щелкните на кнопке Build and Run (кнопка воспроизведения) — и любуйтесь результатом!



Итак, у нас имеются базовые представления и переходы между ними. Теперь можно переходить к работе с реальными данными.



У нас целых три представления... но нет данных! Как заполнить табличное представление и детализированную информацию?

Джо: Контроллер представления TableView знает о табличном представлении, поэтому данные можно сохранить здесь.

Джим: Но одного названия альбома недостаточно. Нам также нужно будет сохранить все, что относится к детализированному представлению.

Джо: Эти данные нужно где-то сохранить...

Фрэнк: ...только этим данным не место в контроллерах представлений.

Джим: Почему нет? Контроллер представления обладает полной информацией о происходящем, так почему бы не сохранить в нем данные?

Фрэнк: А что будет, когда нам потребуется добавить новый альбом или отредактировать альбомы в другом представлении? Получается, что все представления должны обладать информацией о контроллере главного представления?

Джо: Нет, я понимаю, к чему ты клонишь. Кажется, здесь уместно использовать MVC?

Джим: Что такое MVC?

Фрэнк: MVC — паттерн «Модель—Представление—Контроллер». Он постоянно используется в веб-приложениях с такими средами, как Ruby on Rails. Он означает, что визуальная сторона (представление) отделяется от бизнес-логики (контроллер) и от используемых данных (модель).

Джо: Но до настоящего момента мы всегда хранили логику и данные в одном классе.

Фрэнк: Что ж, пора менять ситуацию. Нам потребуется отдельный класс для альбома, а также место для хранения данных.

Джим: Звучит разумно, но какая информация будет храниться в этом классе?

Фрэнк: Необходимо определить поля для данных, построить модель данных...

Паттерн MVC используется для разделения обязанностей...

Паттерн MVC встречается во многих инфраструктурах, а в Socoa Touch он применяется повсеместно. Паттерн MVC требует выделения разных обязанностей в приложении в отдельные классы для сокращения сложности. В разработке iOS граница между представлением и контроллером представления немного размыта, но обычно представление описывается в раскладке, а бизнес-логика содержится в контроллере представления.

Модель представляет данные приложения. Обычно к ней относятся не только собственно данные, но и способ их хранения: подключение к базе данных, веб-служба или простой массив объектов.

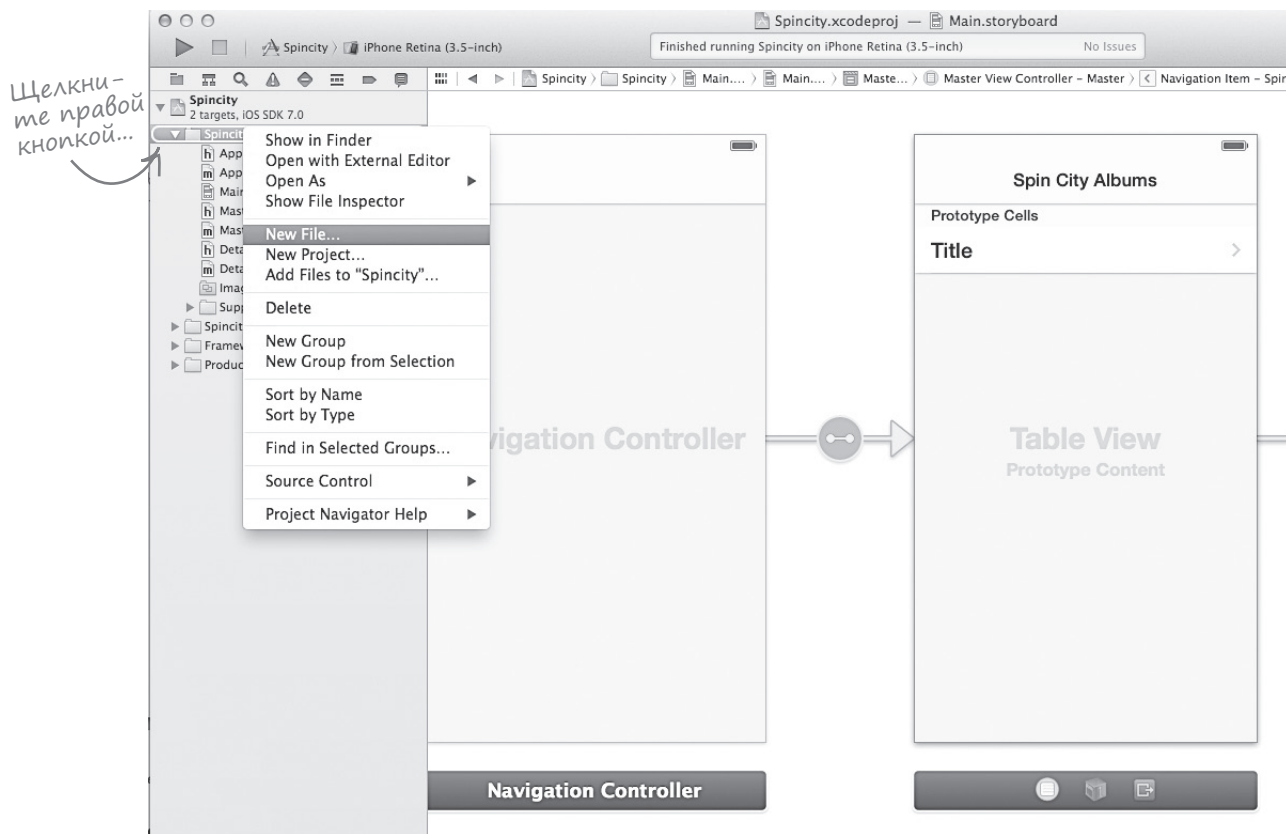


Модель в MVC является концептуальной — в приложении нет единого класса, обеспечивающего функции модели. Приложение может содержать несколько классов, совокупность которых рассматривается как модель. Здесь важно то, что представление данных в приложении, а также механизм работы с ними отделяется от представления и общего управления потоком операций. В приложении SpinCity мы начнем с класса, представляющего альбом, а потом будем постепенно строить приложение.



Добавление нового класса

До сих пор мы работали только с классами, входящими в шаблон, поэтому для добавления нового класса в Xcode необходимо рассмотреть ситуацию более подробно. При создании нового класса среда Xcode генерирует файлы заголовка и реализации с использованием мастера (wizard). Вероятно, в этот момент также будет уместно упомянуть о папках в дереве файлов Xcode. На самом деле это всего лишь группы для упорядочения содержимого Xcode; они вовсе не обязаны как-то влиять на фактическое местонахождение файлов на диске. Мы включим свой новый класс в группу SpinCity.



1

Сгенерируйте класс

Сделайте правый щелчок на папке проекта SpinCity в Xcode и выберите команду **New File...** На экране появляется диалоговое окно для создания нового класса. Убедитесь в том, что в категории **iOS** выбран пункт **Cocoa Touch**; в правой части окна должен присутствовать вариант **Objective-C class**. Выделите его и щелкните на кнопке **Next**. Введите имя класса **Album** и щелкните на кнопке **Next**; сохраните его в проекте SpinCity и группе SpinCity. Убедитесь в том, что в поле **Target** по-прежнему выбран вариант SpinCity. Щелкните на кнопке **Create**.

**НАПРЯГИ МОЗГИ**

Итак, нам понадобится класс, представляющий альбом. Попробуйте составить список свойств, в которых будут храниться нужные сведения. Подумайте, к какому типу должно относиться каждое свойство. Также в классе альбома должен присутствовать один метод. Какой? Если сможете догадаться — дополнительный балл!

Свойства класса



Метод класса
Album

Свойства предоставляют доступ к атрибутам класса

Информация об альбомах, которая должна быть доступна пользователям, хранится в классе Album. Для этого мы объявляем набор свойств, который преобразуется компилятором Objective-C в методы доступа к полям класса. Обновите класс Album и добавьте в него свойства для основных атрибутов: названия альбома, автора, сводного описания, местонахождения в магазине и цены. Также в класс Album будет включен метод для инициализации нового объекта Album, чтобы при создании нового объекта можно было присвоить полям нужные значения.



Упражнение!

Новая модель данных понемногу начинает обретать конкретную форму. Откройте заголовочный файл и файл реализации Album и приведите их к следующему виду.

Обратите внимание: для строк создаются копии, поскольку они являются объектами. Для данных float мы ограничиваемся простым присваиванием, так как это примитивный тип.

Пять свойств данных, которые нам понадобятся.

```
@interface Album
@property (nonatomic, copy) NSString *title;
@property (nonatomic, copy) NSString *artist;
@property (nonatomic, copy) NSString *summary;
@property (nonatomic, copy) NSString *locationInStore;

@property (nonatomic, assign) float price;

-(id)initWithTitle:(NSString *)title artist:(NSString *)
artist summary:(NSString *)summary price:(float)price
locationInStore:(NSString*) locationInStore;
@end
```

Большинство свойств относится к строковому типу, но имеется одно свойство типа float для цены пластинки.

Метод инициализирует только что созданный объект Album. Реализация метода включается в файл .m.



Album.h

```
@implementation Album
```

```
-(id)initWithTitle:(NSString *)title artist:(NSString *) artist
summary:(NSString *)summary price:(float)price locationInStore:(NSString
*)locationInStore {
```

```
    self = [super init];
```

```
    if (self) {
```

```
        _title = title;
```

```
        _artist = artist;
```

```
        _summary = summary;
```

```
        _price = price;
```

```
        _locationInStore = locationInStore;
```

```
        return self;
```

```
    }
    return nil;
```

```
}
@end
```

Мы написали собственный инициализатор, который задает значения свойств на основании переданных аргументов.

Всегда вызывайте инициализатор суперкласса. Если произойдет что-то нежелательное, он вернет nil — как это делаем мы в конце текущего метода.

Инициализаторы — одно из мест, в которых правила Objective-C слегка изменяются. В инициализаторах свойства не используются, вы обращаетесь к переменным экземпляров напрямую. Если предположить, что переменные экземпляров были сгенерированы за вас Objective-C, их имена будут построены по схеме `propertyname`.

Всегда возвращайте self, если инициализация прошла нормально. А если нет — возвращайте nil...



Album.m

Часто задаваемые вопросы

В: Откуда мы возьмем эти данные?

О: На первых порах они будут храниться в массиве, но возможны и другие варианты: списки plist, базы данных SQL и т. д. Практически в любом приложении присутствует некое хранилище данных; мы рассмотрим несколько вариантов организации хранения в книге.

Некоторые источники данных существуют исключительно на базе веб-технологий и ограничиваются локальным кэширова-

нием. Любое приложение, работающее с Dropbox API или iCloud, использует веб-сервисы для получения и отправки данных. Такие сервисы часто используются для обеспечения синхронизации данных между устройствами.

В: Что произойдет, если инициализатор вернет nil?

О: Возврат nil должен сообщить вызывающей стороне, что инициализация завершилась неудачей. Вызывающая сторона должна обработать сбой и обеспечить про-

должение работы. На практике это весьма неприятная ситуация, которая встречается относительно редко. Тем не менее важно соблюдать это правило, так как инфраструктура Cocoa Touch основана на соблюдении этой схемы.

В: Как организовать вывод данных для пользователя?

О: Сначала нужно понять, как получить используемые данные и создать все нужные объекты Album. Давайте пока начнем с этого...



Как организовать загрузку/сохранение данных, которые будут представлены объектами?

Сейчас у нас имеется класс, инкапсулирующий понятие альбома, но мы так и не знаем, откуда будут браться эти данные. Мы уже говорили о том, почему данные не следует хранить в контроллере представления. Тогда где их хранить и как потом до них добраться?

Это распространенная проблема/вопрос в разработке для iOS. И мы вплотную подходим к одному важному паттерну...

Объекты доступа к данным скрывают низкоуровневые операции с данными

По аналогии с тем, как для работы с представлениями используется контроллер, мы используем класс для маскировки фактической выборки данных, сохраняемых в экземплярах нашего класса Album. Скрывая процесс выборки, мы избавляемся от необходимости загромождать контроллер представления кодом SQL или вызовами веб-служб. Обращения к данным инкапсулируются в **Объектах доступа к данным**, или DAO (Data Access Object).

Объекты DAO должны получать запросы вида «Мне нужен альбом 12» и преобразовывать их в запросы, которые работают с данными, извлекают нужную информацию и передают ее контроллеру представления в некотором осмысленном виде.



Возьми в руку карандаш



Итак, вы в общих чертах представляете, какая функциональность должна быть реализована в DAO. Давайте попробуем создать объект DAO для приложения SpinCity. Среди перечисленных ниже вариантов пометьте то, что, по вашему мнению, должно присутствовать в классе AlbumDataController.

☐ Переходы между представлениями.

☐ Предоставление данных по умолчанию для альбомов.

☐ Изображения значков.

☐ Код инициализации альбома при создании.

☐ Действия по нажатию кнопки добавления (+).

☐ Код подсчета альбомов и предоставления текущего значения счетчика контроллерам представления.

Возьми в руку карандаш



Решение

Итак, вы в общих чертах представляете, какая функциональность должна быть реализована в DAO. Давайте попробуем создать объект DAO для приложения SpinCity. Среди перечисленных ниже вариантов пометьте то, что, по вашему мнению, должно присутствовать в классе AlbumDataController.

☐ Переходы между представлениями.

☒ Предоставление данных по умолчанию для альбомов.

☐ Изображения значков.

Размещаются в списках plist приложения

Класс будет содержать массив по умолчанию, заполненный некоторыми исходными данными.

☒ Код инициализации альбома при создании.

☐ Действия по нажатию кнопки добавления (+).

☒ Код подсчета альбомов и предоставления текущего значения счетчика контроллерам представления.

Для заполнения табличного представления необходимо знать, сколько альбомов будет представлено в таблице...



И все это, как и прежде, нужно разместить в заголовочном файле?

На этот раз нет.

Помните: мы намеренно пытаемся абстрагировать (читайте: скрыть) подробности реализации хранения данных. Объекты DAO рассчитаны на две разные категории пользователей. Это внешний мир, который желает получать и сохранять объекты Album, и реализация DAO, которой необходимо до мельчайших подробностей знать, как именно организовано хранение данных. Для этого нам понадобятся два разных интерфейса: **открытый** и **закрытый (приватный)**.

Упражнение!

Создайте новый класс с именем AlbumDataController так же, как это делалось ранее. Обновите файлы заголовка и реализации, чтобы они выглядели так, как показано ниже.

Возвращает объект Album с заданным индексом — при этом мы не раскрываем способ хранения данных, а лишь сообщаем, что пользователь контроллера может осуществлять их выборку по индексу. Выборка может осуществляться по уникальному идентификатору альбома, по местонахождению в магазине, по имени и т. д.

```
#import <Foundation/Foundation.h>
```

```
@class Album;
```

```
@interface AlbumDataController : NSObject
```

```
- (NSUInteger)albumCount;
```

```
- (Album *)albumAtIndex: (NSUInteger) index;
```

```
- (void)addAlbumWithTitle: (NSString *)title artist: (NSString *)artist summary: (NSString *)summary price: (float)price locationInStore: (NSString*)locationInStore;
```

```
@end
```

Опережающее объявление класса Album присутствует здесь, потому что подробная информация о классе Album нам пока не нужна; достаточно знать, что такой класс существует.

Нам понадобится счетчик альбомов, чтобы табличное представление знало, сколько ячеек нужно отобразить в таблице. Обратите внимание: это метод без аргументов, а не свойство.

Создаем новый Album с подходящей информацией и сохраняем его. Вызываем, чему не нужно знать, как это делается, достаточно знать, что это работает.

Включение файла «Album.h» — так как мы собираемся вызывать методы Album в этом классе, необходимо иметь информацию об интерфейсе Album.

AlbumDataController.h

Объявление свойства для массива NSMutableArray, в котором будут храниться данные альбомов (база данных еще не используется). Помните: подробности закрыты от внешнего пользователя, и их можно будет изменить позднее без нарушения работоспособности программы.

```
#import "Album.h"
```

```
@interface AlbumDataController ()
    @property (nonatomic, readonly) NSMutableArray *albumList;
```

```
- (void) initializeDefaultAlbums;
```

```
@end
```

```
@implementation AlbumDataController
```

```
...
```

Добавляем закрытый интерфейс AlbumDataController. Здесь объявляется то, что мы хотим использовать, но не включать в открытый (общедоступный) интерфейс. И здесь будет организовано хранение данных.

Закрытый метод (не входит в открытый интерфейс), который будет использоваться для заполнения массива данными по умолчанию до того, как мы организуем хранение информации в базе данных.

Продолжение на следующей странице...

Упражнение!

```
(id) init {
    self = [super init];

    if (self) {
        _albumList = [[NSMutableArray alloc] init];
        [self initializeDefaultAlbums];
        return self;
    }
    return nil;
}

(void)initializeDefaultAlbums {
    [self addAlbumWithTitle:@"Infected Splinter" artist:@"Boppin'
Beavers" summary:@"Awesome album with a hint of Oak." price:9.99f
locationInStore:@"Section F"];

    [self addAlbumWithTitle:@"Hairy Eyeball" artist:@"Cyclops"
summary:@"A 20/20 retrospective on Classic Rock." price:14.99f
locationInStore:@"Discount Rack"];

    [self addAlbumWithTitle:@"Squish" artist:@"the Bugz" summary:@"Not your
average fly by night band." price:8.99f locationInStore:@"Section A"];

    [self addAlbumWithTitle:@"Acid Fog" artist:@"Josh and
Chuck" summary:@"You should know this stuff." price:11.99f
locationInStore:@"Section 9 3/4"];
}

(void)addAlbumWithTitle:(NSString *)title artist:(NSString *)artist
summary:(NSString *)summary price:(float)price locationInStore:(NSString
*)locationInStore {
    Album *newAlbum = [[Album alloc] initWithTitle:title artist:artist
summary:summary price:price locationInStore:locationInStore];

    [self.albumList addObject:newAlbum];
}

(NSUInteger)albumCount {
    return [self.albumList count];
}

(Album *)albumAtIndex:(NSUInteger)index {
    return [self.albumList objectAtIndex:index];
}

@end
```

Обновите метод `init`, чтобы в нем создавался массив для хранения данных, а при создании контроллера данных вызывался наш новый метод `initializeDefaultAlbums`.

Реализация метода `initializeDefaultAlbums` должна использовать открытый метод `addAlbumWith...` для создания нескольких альбомов по умолчанию.

Реализуйте метод `addAlbumWith...` для создания нового объекта альбома по заданной информации и включения его в массив.

Реализации методов `albumCount` и `albumAtIndex` просто используют базовый массив с данными.



AlbumDataController.m



Поздравляем — вы только что создали свой первый объект DAO!

Осталось лишь убедиться в том, что Xcode не выдает предупреждений и ошибок. Впрочем, ничего интересного в приложении вы еще не увидите — нужно связать его с табличным представлением. Пока же нам предстоит немалая работа.

Часть Задаваемые Вопросы

В: Зачем все это было нужно? Почему нельзя было просто добавить массив в контроллер представления?

О: Можно, но у контроллера представления уже есть свои дела. В iOS контроллеры представлений нередко превращаются в «свалку» для кода, которому там не место. Мы воспользовались стандартным паттерном, который упростит последующие изменения, если в будущем мы захотим добавить в приложение поддержку базы данных или Core Data.

Использование паттерна DAO существенно упрощает сопровождение кода и соответствует коду Apple, который вы можете загрузить при просмотре демонстрационных приложений.

В: Я все еще плохо понимаю, зачем нужно это деление на открытый и закрытый интерфейсы. Почему мы поместили директиву @interface в файл реализации?

О: Objective-C использует ключевое слово @interface для определения места, в котором объявляются свойства и методы класса. Обычно @interface используется в заголовочном файле для объявления открытого интерфейса класса. Другие классы включают заголовочный файл и знают, что им может предложить этот класс. Однако часто встречаются ситуации, в которых требуется создать внутренние свойства и методы, используемые только внутренней реализацией и не предназначенные для посторонних. При этом все равно удобно использовать синтаксис свойств; просто вы не хотите, чтобы посторонние лезли во внутреннюю реализацию. Для этого в файле реализации объявляется закрытый интерфейс, который создается включением в файл реализации секции @interface. Поскольку этот файл видим только непосредственной реализации класса, только этому коду известно о существовании интерфейса.

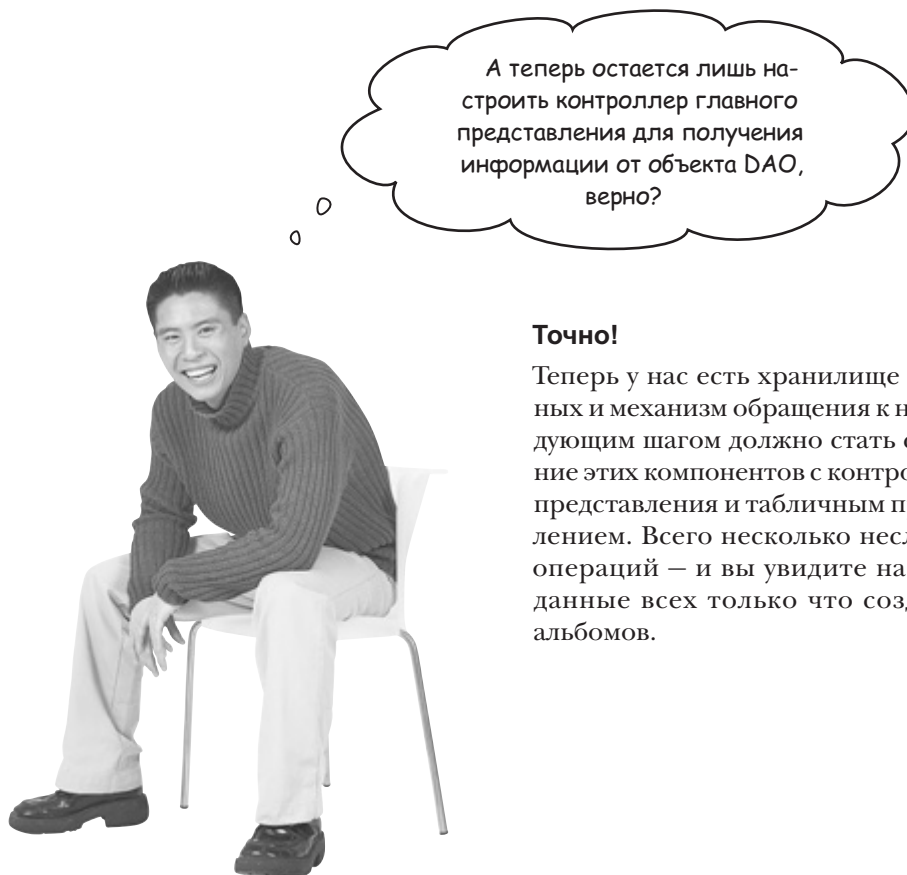
В: И никакой другой код не сможет вызывать мои закрытые методы?

О: Более или менее правильно. Так как закрытые методы не объявляются в заголовочном файле, другой код не должен знать об их существовании. Objective-C поддерживает динамический вызов методов посредством передачи сообщений и не мешает стороннему коду проанализировать ваш класс во время выполнения и отправить ему сообщение, на которое ваш класс отреагирует. Компилятор выдаст грозное предупреждение, которое напугает слабонервного нарушителя, но технически такая возможность существует.

В: Значит, вот что имеет в виду Apple, когда предлагает «не использовать закрытые API»?

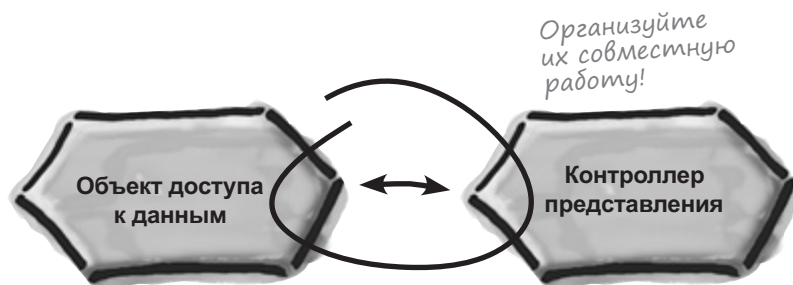
О: Да! Именно. Попробуйте поэкспериментировать с вызовом закрытых методов для классов CocoaTouch, и ваше приложение моментально вылетит из App Store!

Переходим к связыванию объекта DAO с представлением!



Точно!

Теперь у нас есть хранилище для данных и механизм обращения к ним. Следующим шагом должно стать связывание этих компонентов с контроллером представления и табличным представлением. Всего несколько несложных операций — и вы увидите на экране данные всех только что созданных альбомов.





Упражнение

Настройте `MasterViewController` для работы с классом `AlbumDataController`. Разумеется, в шаблонном коде эта настройка не выполняется, так что придется внести пару изменений...

1

Удалите часть шаблонного кода из `MasterViewController.m`.

Наше приложение выходит за рамки простейшего шаблона, поэтому шаблонный код придется отредактировать. Удалите объявление `NSMutableArray` в закрытом интерфейсе, создание кнопки из `viewDidLoad`, методы `insertNewObject` и `commitEditingStyle`. Наконец, измените метод `canEditRowAtIndexPath` так, чтобы он возвращал `NO` вместо `YES`.

2

Добавьте код для работы с `AlbumDataController` в `MasterViewController.m`.

Добавьте директивы импортирования классов `AlbumDataController` и `Album` в начало файла. Объявите свойство для хранения `AlbumDataController` и инициализируйте свойство `AlbumDataController` в `awakeFromNib`.



Настройте MasterViewController для работы с классом AlbumDataController. Разумеется, в шаблонном коде эта настройка не выполняется, так что придется внести пару изменений...

1

Удалите часть шаблонного кода из MasterViewController.m.

Наше приложение выходит за рамки простейшего шаблона, поэтому шаблонный код придется отредактировать. Удалите объявление NSMutableArray в закрытом интерфейсе, создание кнопки из viewDidLoad, методы insertNewObject и commitEditingStyle. Наконец, измените метод canEditRowAtIndexPath так, чтобы он возвращал NO вместо YES.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.navigationItem.leftBarButtonItem = self.editButtonItem;
    UIBarButtonItem *addButton = [[UIBarButtonItem alloc] initWithBarButtonItemSystemItem:UIBarButtonItemSystemItemAdd target:self action:@selector(insertNewObject:)];
    self.navigationItem.rightBarButtonItem = addButton;
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}

- (void)insertNewObject:(id)sender
{
    if (!_objects) {
        _objects = [[NSMutableArray alloc] init];
    }
    [_objects insertObject:[NSDate date] atIndex:0];
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:0 inSection:0];
    [self.tableView insertRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationAutomatic];
}
```



MasterViewController.m

```

- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath
{
    return NO;
}

- (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        [_objects removeObjectAtIndex:indexPath.row];
        [tableView deleteRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationFade];
    } else if (editingStyle == UITableViewCellEditingStyleInsert) {
        //
    }
}

```



MasterViewController.m

2

Добавьте код для работы с AlbumDataController в MasterViewController.m.

Добавьте директивы импортирования классов AlbumDataController и Album в начало файла. Объявите свойство для хранения AlbumDataController и инициализируйте свойство AlbumDataController в awakeFromNib.

```

#import "MasterViewController.h"
#import "DetailViewController.h"
#import "AlbumDataController.h"
#import "Album.h"

```

Ссылка на объект AlbumDataController является сильной (strong); это означает, что она не будет уничтожена даже при выгрузке представления.

```

@interface MasterViewController ()
    @property (nonatomic, strong) AlbumDataController *albumDataController;
@end

```



MasterViewController.m



Упражнение
Решение

Настройте MasterViewController для работы с классом AlbumDataController. Разумеется, в шаблонном коде эта настройка не выполняется, так что придется внести пару изменений...

```
@implementation MasterViewController
```

```
-(void)awakeFromNib
```

```
{
```

```
    [super awakeFromNib];
```

```
    self.albumDataController = [[AlbumDataController alloc] init];
```

```
}
```

Здесь можно провести разовую инициализацию при первом создании представления. Учтите, что представление на экране еще не отображается.



MasterViewController.m



Вперед!!

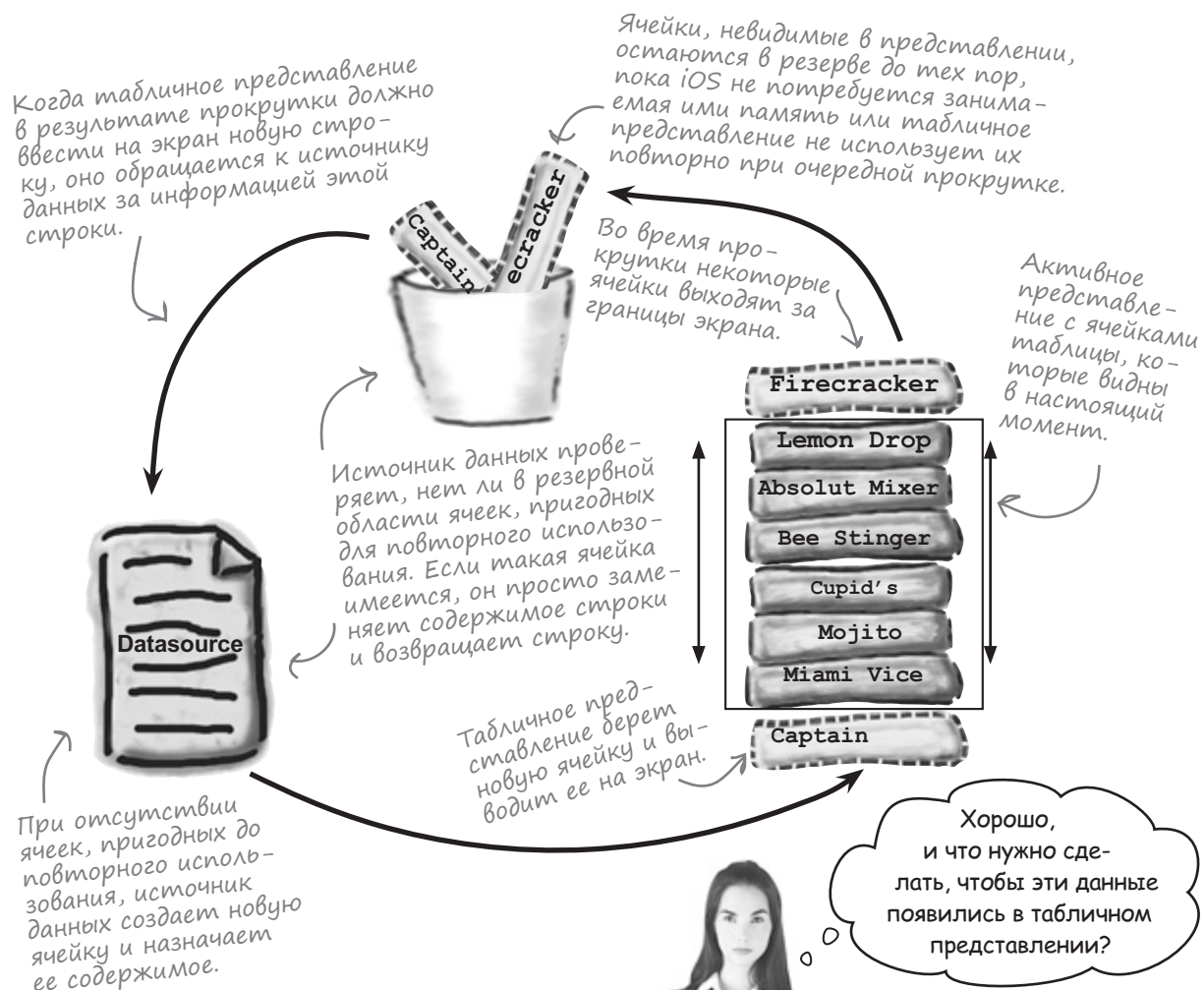
Все готово, можно ехать!

Данные и логика работы с ними расставлены по местам. Садитесь за руль — пора заставить табличное представление поработать.

Таблица состоит из ячеек

Теперь, когда вы знаете, как получить доступ к объектам Album, нужно как-то вывести их на экран. Для этого мы воспользуемся табличным представлением. В настоящий момент у нас имеется лишь несколько объектов Album, созданных для примера, но при работе с полным каталогом магазина придется иметь дело с сотнями и даже тысячами альбомов. Вместо того чтобы загружать все данные и пытаться их втиснуть в таблицу одновременно, табличное представление оптимизирует использование памяти: если ячейка находится за пределами экрана, для пользователя совершенно неважно, содержит она данные или нет.

Компоненты UITableView отображают столько данных, сколько необходимо для заполнения экрана iPhone или представления в iPad — и не важно, сколько данных у вас может быть на самом деле. Для этого UITableView эффективно использует ячейки, находящиеся за пределами экрана.





Код ячейки таблицы под увеличительным стеклом

Методы табличного представления находятся в классе `MasterViewController` (что вполне разумно, поскольку главное представление отвечает за табличное). Код шаблона содержит методы для табличных представлений, а также дополнительный код (закомментированный) для переупорядочивания ячеек. Наше приложение предоставляет доступ к данным только для чтения, так что этот код нам не понадобится.

```
#pragma mark - Table View
```

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
```

```
{
```

```
    return 1;
```

```
}
```

Методы сообщают табличному представлению, сколько секций содержит таблица и сколько строк в каждой секции.

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
```

```
{
```

```
    return [self.albumDataController albumCount];
```

```
}
```

Обновите табличное представление, чтобы получить количество строк от `albumDataController`

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
```

```
{
```

`indexPath` содержит представление секции и количество строк в необходимой ячейке.

```
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
```

```
@AlbumCell forIndexPath:indexPath];
```

```
    Album *album = [self.albumDataController albumAtIndex:indexPath.row];
```

```
    NSDate *object = _objects[indexPath.row];
```

```
    cell.textLabel.text = album.title;
```

```
    return cell;
```

```
}
```

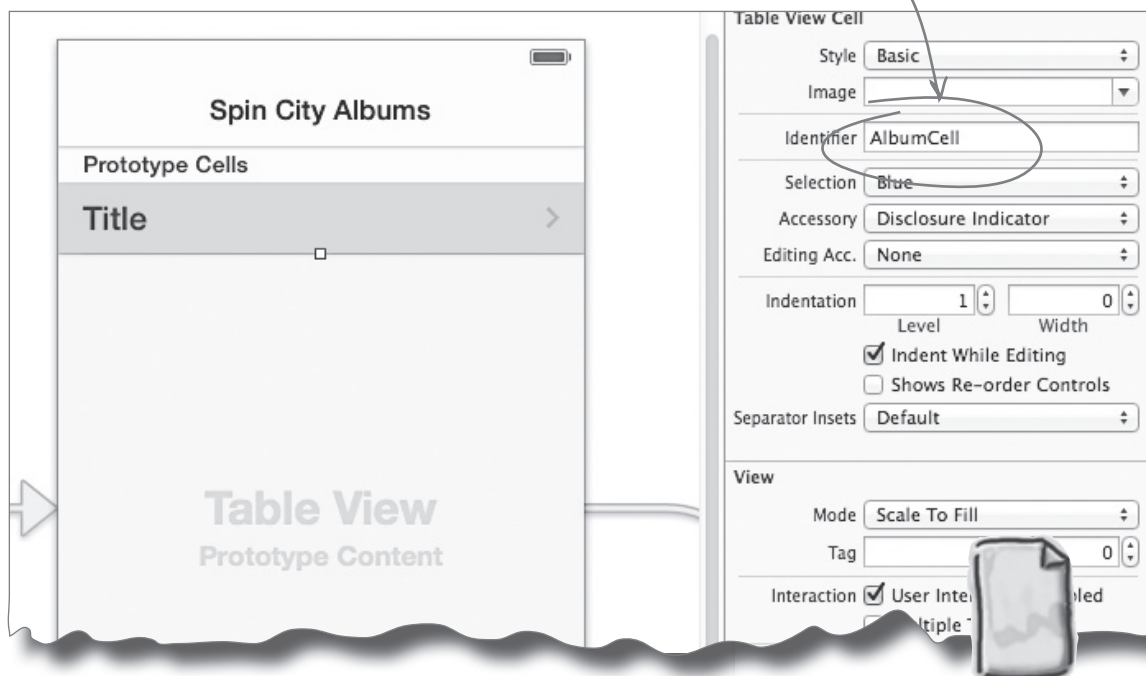
Текст ячейки обновляется информацией о конкретном альбоме, который нам нужно вывести на экран.

Идентификатор `AlbumCell` связывает ячейку с кодом, поэтому ему необходимо присвоить содержательное имя. Это имя необходимо изменить здесь и в раскадровке.



MasterViewController.m

Вторая часть назначения
имени ячейкам таблицы...



MainStoryboard.storyboard

```

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"showDetail"]) {
        NSIndexPath *indexPath = [self.tableView indexPathForSelectedRow];
        Album *album = [self.albumDataController albumAtIndex:indexPath.row];
        NSDate *object = _objects[indexPath.row];
        [[segue destinationViewController] setDetailItem:album];
    }
}

```

Этот код получает выбранный альбом и передает данные детализированному представлению, которое должно появиться на экране.



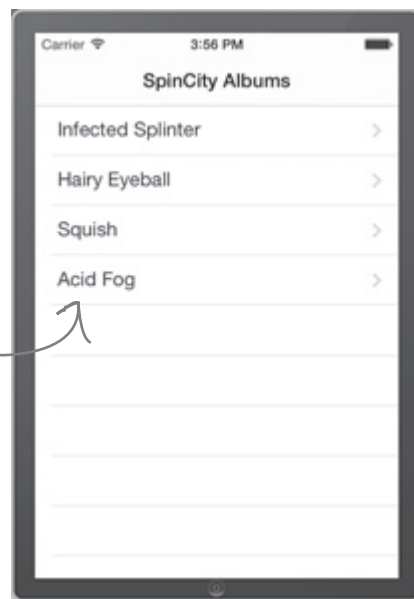
MasterViewController.m



ТЕСТ-ДРАЙВ

После того как табличное представление будет связано с данными, можно запустить приложение и просмотреть альбомы, которые были помещены в табличное представление через DAO. Если хотите, добавьте в DAO еще несколько альбомов!

Убедитесь сами — список еще и прокручивается!



Часто Задаваемые Вопросы

В: Для чего нужны методы, которые мы удалили из `MasterViewController`?

О: Класс `MasterViewController` наследует от `UITableViewController` — встроенного класса для работы с табличными представлениями. Табличное представление настроено для использования нашего контроллера представления как источника данных и делегата; это означает, что когда в программе что-то происходит, табличное представление будет вызывать методы контроллера представления.

В исходном виде шаблонный контроллер представления поддерживает размещение кнопки + для добавления новых элементов, а также возможность удаления и перестановки элементов списка. В на-

шем приложении такая функция отсутствует, поэтому мы удалили шаблонные методы. Полный перечень доступных методов делегата приведен в описании протокола `UITableViewDelegate` в документации iOS.

В: А что происходит с путями?

О: Помните, как в раскладке две сцены соединялись активным путем (переходом)? Шаблонный код настроен так, чтобы при прикосновении к строке инициировался этот путь. Он выполняет необходимую подготовку следующего представления, после чего обращается с обратным вызовом к контроллеру представления, уведомляя его о готовящемся переходе (вызов `prepareForSegue`, который мы заполнили кодом). В методе `prepareForSegue` именно это и происходит: мы готовимся к появлению

следующего представления. Иногда в нем закрываются сетевые подключения или выполняются другие завершающие действия для уходящего представления. В других случаях (как в нашем) необходимо сообщить новому представлению, которое собирается появиться на экране, о происходящем. Мы просто берем новое представление из пути и сообщаем ему, к какому альбому прикоснулся пользователь. Новое представление открывает доступ к методу `setDetailItem` (через свойство). Если бы с новым представлением, готовым появиться на экране, нужно было выполнить какие-то дополнительные действия, это тоже можно было бы проделать здесь. Обратите внимание на проверку пути, выполняющего переход, в коде метода — может оказаться, что из представления выходят сразу несколько путей, ведущих в разные места.

Сюрприз

Squish

В принципе неплохо, но для меня в табличном представлении маловато информации... Нельзя ли вывести в ячейке более подробные сведения?



Конечно!

Обговорив все с Робом, мы решили, что в ячейке табличного представления также должно выводиться краткое описание:

Что я хочу видеть на первом экране:
Название альбома
Краткое описание
Цена

Возьми в руку карандаш



Постройте новый макет ячейки, руководствуясь тем, какую информацию Роб хочет видеть в каждой ячейке главного представления своего приложения.

Неправильных ответов здесь быть не может...



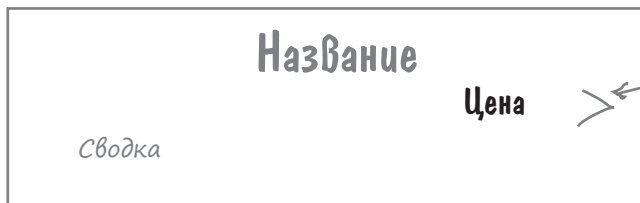
И помните, что можно использовать более одного столбца...

Возьми в руку карандаш



Решение

Самым заметным элементом ячейки должно быть название альбома; под ним выводится краткое описание.



Здесь по правилам Apple должен находиться индикатор вывода дополнительной информации...

Вот как выглядит макет ячейки для вывода дополнительной информации. В таких ситуациях стоит привлечь к работе профессионального дизайнера. Информации должно быть много, чтобы она приносила реальную пользу, но не загромождала макет, затрудняя работу с таблицей.

Раскадровка может пригодиться для макетов с нестандартными ячейками табличных представлений

До настоящего момента мы использовали стандартные макеты ячеек табличных представлений. Они работают нормально, но этого мало. У ячеек табличных представлений существует пара стилевых модификаций со свойствами `detailTextLabel` и `imageView`. Если они подойдут для вашего приложения, используйте их. А мы покажем, как создать нестандартную ячейку, это не трудно, и вы можете делать со своими ячейками все, что угодно.

Так как мы не будем использовать ячейки по умолчанию, нам придется создать новый класс, производный от базового `UITableViewCell`, и добавить дополнительные свойства, которые должны отображаться на экране.

Начните с этого варианта и опробуйте разные типы ячеек. Ни один из стандартных вариантов нам не подойдет.

Код ячейки таблицы по умолчанию.

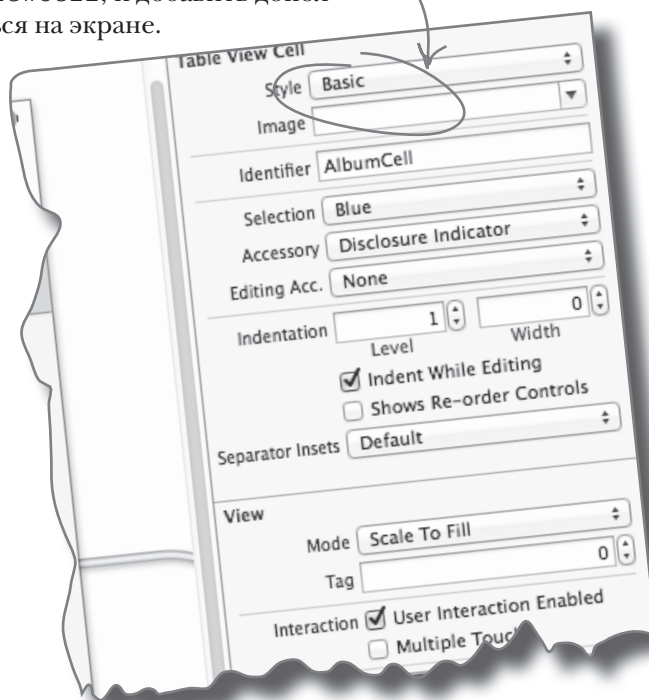


UITableViewCell

Код нестандартной ячейки таблицы.



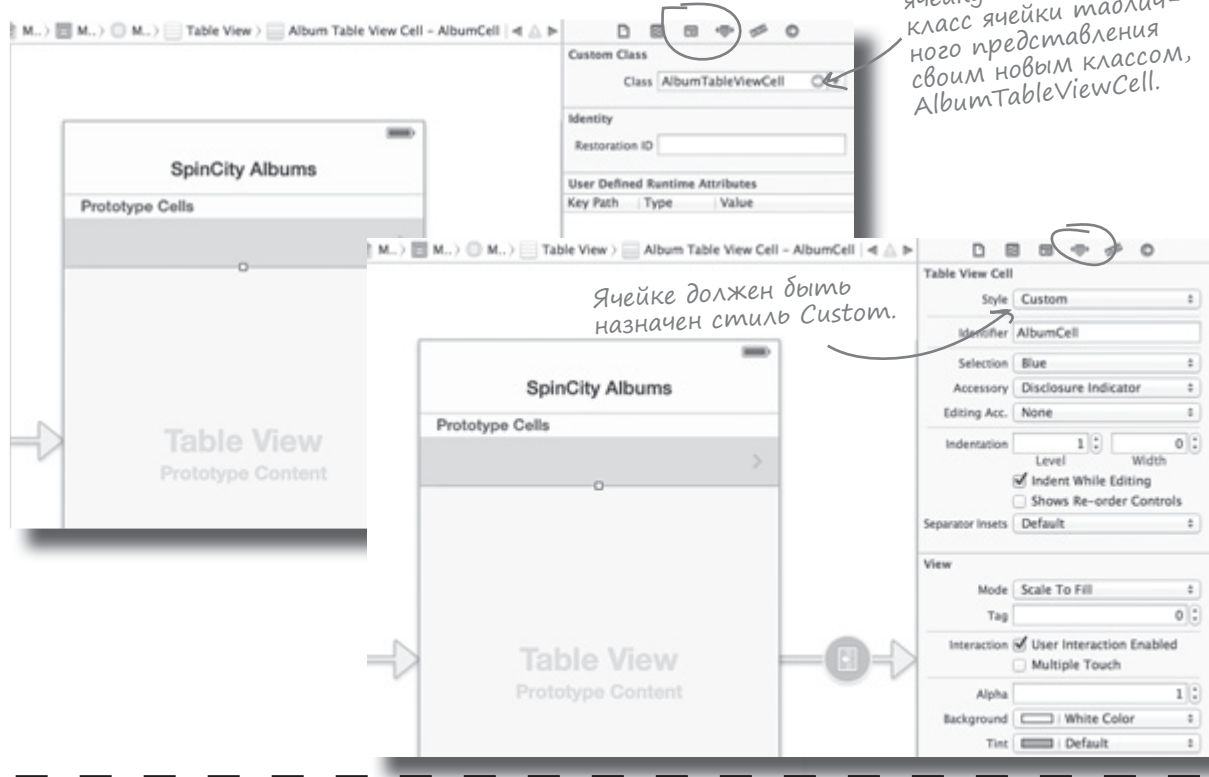
AlbumTableViewCell



ПОСТРОЕНИЕ НЕСТАНДАРТНОЙ ЯЧЕЙКИ ТАБЛИЦЫ

Чтобы построить новую ячейку таблицы, необходимо создать новый класс, производный от класса `UITableViewCell`.

- 1 Создайте новый класс Objective-C с именем `AlbumTableViewCell`, производный от `UITableViewCell`. Проследите за тем, чтобы он находился в группе `SpinCity` и чтобы проект `SpinCity` был выбран в качестве целевого.
- 2 Затем следует изменить раскраску, чтобы в ней использовалась новая нестандартная ячейка (вместо ячейки по умолчанию). Вернитесь к файлу `MainStoryboard.storyboard` и щелкните на пустой ячейке табличного представления в сцене `Table View`.



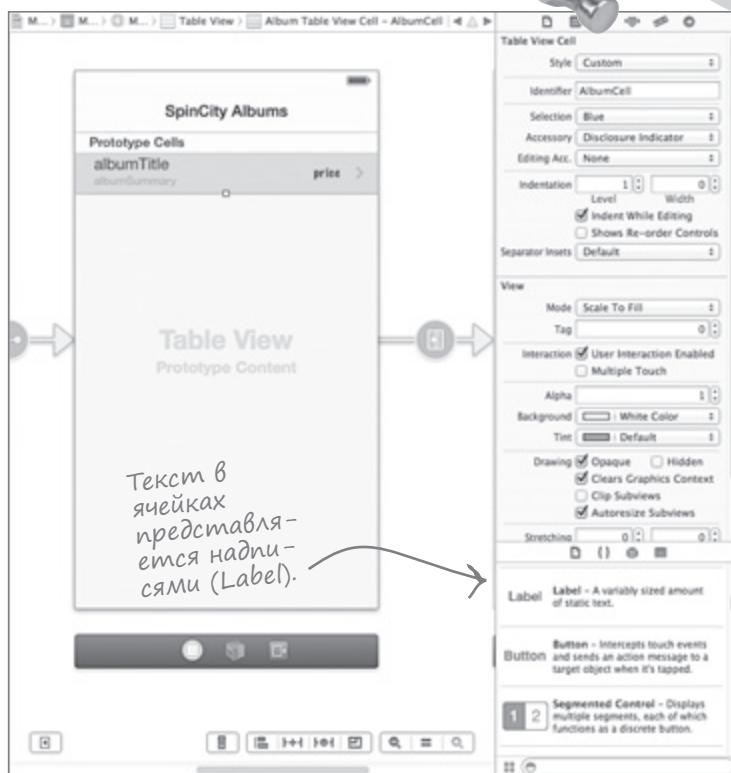
ПОСТРОЕНИЕ НЕСТАНДАРТНОЙ ЯЧЕЙКИ ТАБЛИЦЫ (ПРОДОЛЖЕНИЕ)

Теперь для создания макета с новыми ячейками нам предстоит работать в раскладовке и перетаскивать элементы на ячейку, как мы это делаем с элементами представлений.

- 3 Перетащите надпись для названия альбома. Разместите в верхней половине ячейки с рекомендованным левым отступом. Установите ширину равной приблизительно половине ячейки. Замените текст по умолчанию на `albumTitle`.

- 4 Перетащите вторую надпись, предназначенную для краткого описания. Разместите ее в нижней половине ячейки с рекомендуемым левым отступом и задайте ширину в $3/4$ от ширины ячейки. В инспекторе атрибутов выберите 12 пт и светло-серый цвет. Замените текст по умолчанию на `albumSummary`.

- 5 Надпись для вывода цены разместите по центру ячейки с рекомендуемым правым отступом. Выберите шрифт Marker Felt Thin, 14 пт, и режим выравнивания текста по правому краю. Растяните надпись влево до правого края надписи с кратким описанием. Замените текст по умолчанию на `price`.





ТЕСТ-ДРАЙВ

Макет ячеек построен, осталось лишь связать надписи со ссылками IBOutlet и действиями. Сделайте это, и вы сможете увидеть новые ячейки в действии!

- 1 **Откройте Assistant Editor и выберите файл AlbumTableViewCell.h.**
Раскадровка должна располагаться рядом с представлением Assistant Editor. Перетащите указатель мыши с нажатой клавишей Control с надписи albumTitle в позицию между директивами @interface и @end в файле AlbumTableViewCell.h. Убедитесь в том, что для связи выбран тип Outlet. Введите имя albumtitleLabel, выберите тип UILabel и слабую (weak) ссылку.
- 2 **Повторите процесс для надписей albumSummaryLabel и priceLabel.**
Когда все будет сделано, на полях рядом с файлом AlbumTableViewCell.h должны появиться три заполненных кружка.
- 3 **Откройте файл MasterViewController.m.**
Добавьте директиву импортирования файла «AlbumTableViewCell.h» и обновите метод cellForRowAtIndexPath, чтобы в нем использовался новый тип нестандартной ячейки.

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath: (NSIndexPath *)indexPath
{
    AlbumTableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"AlbumCell" forIndexPath:indexPath];

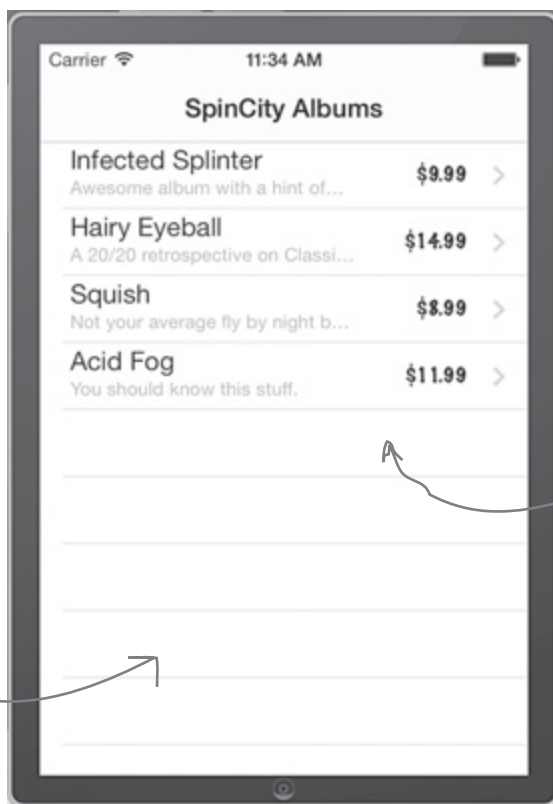
    Album *album = [self.albumDataController albumAtIndex: indexPath.row];
    cell.albumtitleLabel.text = album.title;
    cell.albumSummaryLabel.text = album.summary;
    cell.priceLabel.text = [NSString stringWithFormat:@"%01.2f", album.price];
    return cell;
}
```

- 4 **Запустите приложение....**



ТЕСТ-ДРАЙВ

Работаем!



Список про-
кручивается,
и все такое.

Обратите
внимание на
нестандарт-
ные ячейки
таблицы...



Мне нравится, но когда я прикасаюсь к строке, детализированная информация не выводится!

Мы знаем...

Итак, табличное представление заработало, все идет хорошо, и мы можем взяться за детализированное представление в следующей главе...



Ваш инструментарий представлений

Глава 3 осталась позади, а ваш инструментарий пополнился табличными представлениями.

Множественные представления

Во многих приложениях используются сразу несколько представлений.

Табличные представления хорошо работают в сочетании с детализированными для просмотра иерархических данных.

Навигационный контроллер обеспечивает переходы между представлениями.

Раскадровки

Используются для редактирования всех представлений вашего приложения и переходов между ними.

Здесь также могут редактироваться нестандартные ячейки табличных представлений.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Приложения iOS основаны на прикосновениях к сенсорному экрану — и они были первыми в этой категории.
- MVC — главный паттерн проектирования в приложениях iOS.
- Объекты доступа к данным (DAO, Data Access Objects) используются для сокрытия низкоуровневого доступа к данным.
- Ячейками табличных представлений управляют объекты UITableView.

4 приложения с несколькими представлениями

Такие важные подробности

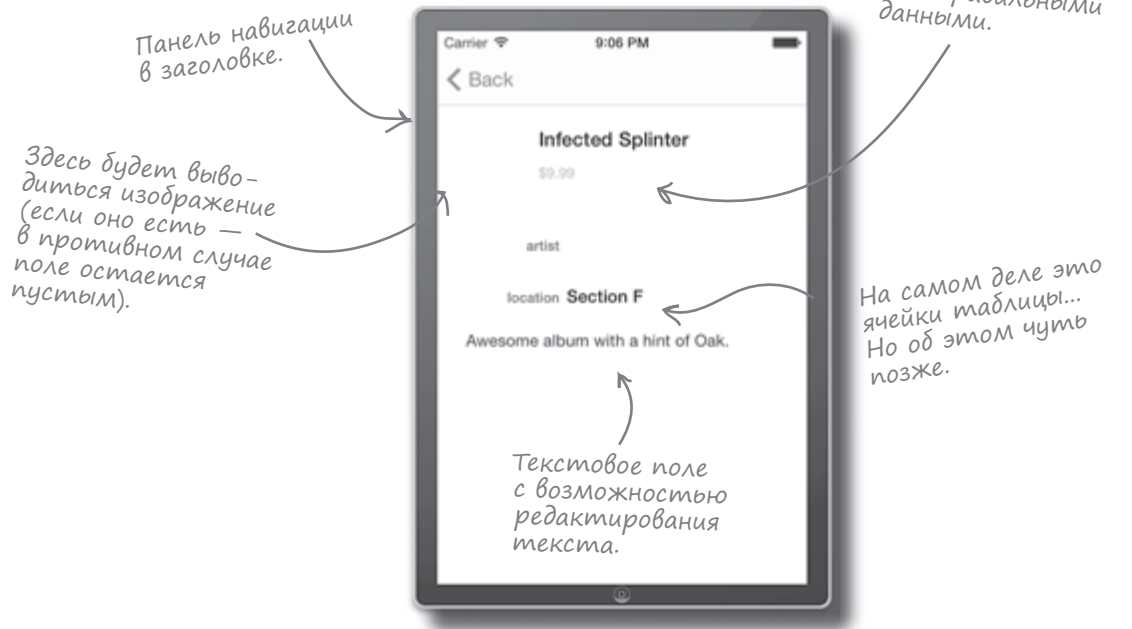
Подробности бывают очень важными — скажем, где стоят деревья...



В большинстве приложений iOS используется более одного представления. Работа над приложением началась резво: мы воспользовались встроенными шаблонами и внесли довольно впечатляющие изменения в табличное представление. Пришло время заняться выводом подробной информации, подготовкой нового представления и управлением навигацией между ними. Многие популярные приложения в App Store предоставляют удобные и простые средства для работы с большими объемами данных. Приложение Spin City делает то же самое — оно помогает пользователю найти нужную пластинку, не перерывая многочисленные конверты на полках!

Детализированные представления

Приложение SpinCity работает и рисует симпатичную таблицу с данными. Теперь, когда мы видим высокоуровневую сводку, пора заняться детализацией. На рисунке изображено детализированное представление, которое мы предложили ранее.



Детализированные представления повсюду

Детализированные представления используются во многих приложениях, и каждый, кто хоть раз пользовался ими, примерно представляет, для чего они нужны. Детализированные представления обычно находятся на нижнем уровне иерархического стека представлений; это означает, что в них выводится самая подробная информация, которую только может предоставить приложение.

Детализированные представления также используются для перевода приложения в другой режим работы (представьте, что вы выходите из поставки новостей и переходите к стене в своем приложении Facebook). В последнее время приложения становятся более сложными, поэтому в них могут появиться дополнительные уровни взаимодействия, для обращения к которым могут использоваться разные детализированные представления.

В нашем приложении табличное представление содержит довольно подробную информацию об альбоме, и это хорошо. И все же нужно предусмотреть возможность вывода еще более полной информации, чтобы покупатель мог узнать о сокровищах Роба все, что душа пожелает.



В основном это надписи и текстовые поля, верно? Как разместить их так, чтобы они хорошо смотрелись? А если название альбома окажется слишком длинным?

Всегда старайтесь найти подходящее встроенное представление.

Макет можно построить вручную, и он будет работать, но существует и более эффективный способ — особенно при работе с текстовой информацией.

Табличные представления можно настроить так, чтобы макет выглядел более привлекательно, и при этом сохранить все преимущества табличной структуры. После предыдущей главы вы можете считать себя настоящим профессионалом, так что с настройкой табличного представления особых проблем быть не должно.



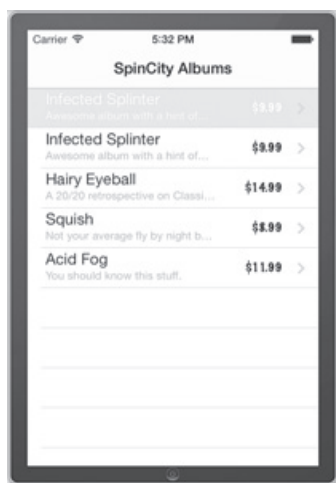
МОЗГОВОЙ ШТУРМ

Проверьте несколько приложений на iPhone или iPad. Сможете ли вы определить, какие из них используют табличные представления в детализированных данных, а какие нет?

Табличные представления не всегда похожи на... таблицы

Вы уже видели пример использования табличного представления в главном представлении. Таблицы хорошо подходят для построения аккуратных макетов (как при использовании таблиц в текстовых документах или электронных таблицах), при условии, что пользователь не может выделять данные в таблице.

Например, детализированные представления в приложении Почта или Контакты не похожи на таблицы, и все же являются ими...



*Но таблицы не всегда
выглядят так!*



Часто Задаваемые Вопросы

В: Что нам дает использование табличного представления вместо определения макета в детализированном представлении?

О: Удобство построения макета и удобные встроенные инструменты. Табличное представление уже включает прокрутку и встроенную навигацию; кроме того, позднее вы сможете подключить к приложению новые представления без особых хлопот.

В: Часто ли приложения поступают так? И что об этом думает Apple?

О: Постоянно! Если вы просто продолжите работать с привычными приложениями, то вы увидите, что это решение более характерно для бизнес-приложений и офисных приложений, чем для нестандартных представлений. Apple также очень часто использует его в своих приложениях...



Джо: Ладно, теперь нужно создать детализированное представление. С чего начать?

Фрэнк: Видимо, нужно вернуться к редактору раскадровки?

Джим: А что потом? Новое представление должно быть табличным. А шаблон нам предлагает обычный класс `DetailViewController`.

Джо: Значит, нужно заменить его, как мы это делали для ячеек таблицы. Как ты думаешь, мы сможем просто воспользоваться контроллером табличного представления из `UIKit`?

Фрэнк: Мы изменили табличное представление по умолчанию, чтобы главное представление выглядело так, как нужно нам. Значит, придется создавать новый класс.

Джим: Верно, но тогда мы субклассируем контроллер табличного представления по умолчанию и получим всю встроенную поддержку, а также возможность настройки макета?

Фрэнк: Точно. И мы можем построить собственную раскадровку для управления макетом.

Джо: А как насчет отображения данных?

Джим: У меня есть кое-какие мысли относительно того, как это сделать. Скажу, когда дойдет до этого.

Джо: Будь по-твоему. Итак, берем файл раскадровки...

Замена UIViewController на UITableView Controller

Так как шаблон главного представления поставляется с простейшим детализированным представлением, нам нужно изменить класс этого представления, чтобы автоматически получить все удобства табличных представлений. Как и в случае с ячейками таблицы, нужные действия выполняются в инспекторе атрибутов.

```
#import <UIKit/UIKit.h>
```

```
@class Album;
```

```
@interface DetailViewController : UIViewController UITableViewController
```

```
@property (strong, nonatomic) id Album *detailItem;
```

```
@property (weak, nonatomic) IBOutlet UILabel *detailDescriptionLabel;
```

```
@end
```

Полностью удалите надпись.

Объявите класс Album за пределами интерфейса.

Замените обобщенный контроллер представления классом UITableViewController.

Замените обобщенный идентификатор id на Album.



DetailViewController.h

```
- (void)configureView
```

```
{
```

```
// Update the user interface for the detail item.
```

```
if (self.detailItem) {
```

```
self.detailDescriptionLabel.text = [self.detailItem description];
```

```
}
```

```
}
```

Удалите эту строку, так как надписи больше нет.



DetailViewController.m

★ КТО И ЧТО ДЕЛАЕТ? ★

Соедините каждый контроллер с описанием того, что он делает.

Контроллер

Что делает

UIViewController

Предоставляет стек навигации для переходов между представлениями; встраивает панель навигации в верхней части представления.

UINavigationController

Контроллер хорошо подходит для приложений с несколькими равноправными представлениями, чтобы пользователь мог легко переключаться между ними.

UITableViewController

Контроллер включает основные средства, необходимые для работы с представлениями.

UITabBarController

Предоставляет системный интерфейс для редактирования видеоданных прямо на устройстве.

UIPageViewController

Встраивает табличное представление в используемое представление. Может использоваться для детализированных представлений или более традиционного табличного представления.

UIVideoEditorController

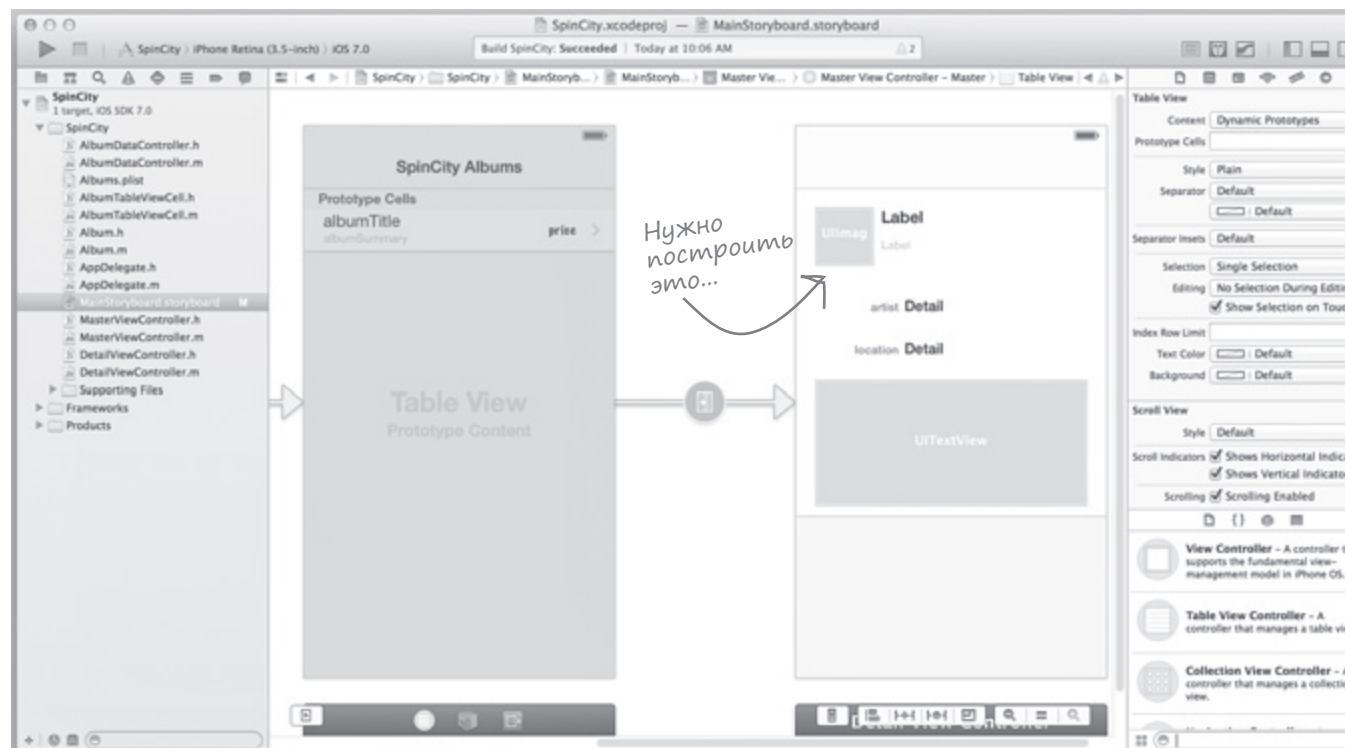
Контроллер используется для обработки перемещений между контроллерами представлений с заданным переходом.

→ Ответы на стр. 196.

Макет нового детализированного представления

Итак, наше приложение знает, что оно взаимодействует с `UITableViewController`, а не с `UIViewController`. Теперь на основании имеющегося эскиза мы займемся настройкой элементов представления в раскладовке.

- ☐ Замените детализированное представление табличным представлением в раскладовке.
- ☐ Обеспечьте выполнение перехода между новым детализированным представлением и табличным представлением.
- ☐ Постройте макет представления в соответствии с эскизом.



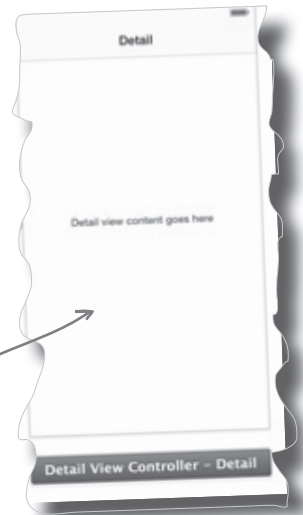
Включение представления в раскладовку

Это нетривиальная часть работы, поэтому она будет описана весьма подробно. Здесь мы отходим от шаблона, поэтому стоит более подробно поговорить как о самом представлении, так и о том, как к нему перейти. А начать следует с того, что мы уже делали в коде ранее.

- 1 В файле `Main.storyboard` удалите всю сцену `Detail View Controller`.

Убедитесь в том, что в Xcode выделено все детализированное представление. Удалите его полностью.

Убедитесь в том, что представление выделено. Нажмите клавишу Delete.



- 2 Перетащите контроллер табличного представления (`Table View Controller`) на раскладовку, чтобы заменить детализированное представление.

При попытке выделить окно выделяется представление, поэтому для выделения контроллера представления следует выделить черное поле в нижней части.



Возьми в руку карандаш



Какой класс следует выбрать для нового представления?

- ☐ MasterViewController
- ☐ DetailViewController
- ☐ UITableViewController

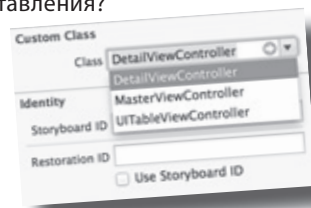
Почему?

Возьми в руку карандаш

Решение

Какой класс следует выбрать для нового представления?

- ☐ MasterViewController
- ☒ DetailViewController
- ☐ UITableViewController



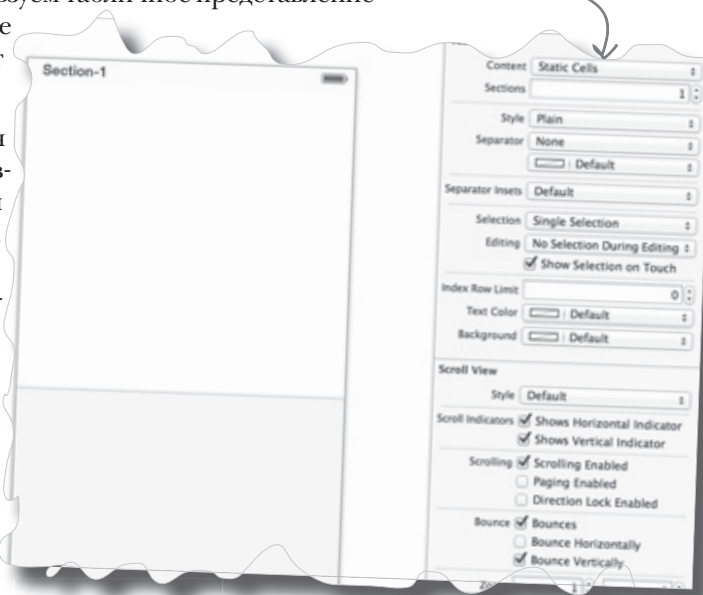
Почему? Потому что это представление одновременно является и детализированным, и табличным. На уровне класса его роль детализированного представления важнее роли табличного представления.

Остальная часть представления строится динамически

Теперь нужно сформировать макет нового табличного представления. В отличие от последнего табличного представления, в котором нужно количество строк было неизвестно, на этот раз мы точно знаем, какая информация будет выводиться, и используем табличное представление для упрощения работы. В данном случае вместо динамической таблицы будет использоваться статическая таблица.

Динамическая таблица используется по умолчанию в табличных представлениях тогда, когда макет таблицы изменяется в зависимости от данных. У нас же входная информация остается неизменной, поэтому макет представления должен оставаться постоянным, то есть статичным.

После выделения табличного представления выберете в поле Content значение «Static Cells».



Для Любопытных

Мы также изменим тип разделителя ячеек. Не забудьте, что мы пытаемся использовать табличное представление как детализированное. Обычно ячейки таблицы разделены одиночной линией, и после ее удаления представления станут неотличимы.

В инспекторе, раздел Separator Insets.

Выделите верхнюю ячейку табличного представления; перетащите границу или воспользуйтесь инспектором, чтобы установить высоту ячейки равной 100 пунктам.

Перетащите надпись в ячейку. Используя рекомендации, разместите ее у правого края представления UIImageView и растяните вправо до границы ячейки. Выверните ячейку с рекомендуемым верхним отступом. В инспекторе атрибутов выберите шрифт System Bold.

Выделите вторую и третью ячейки таблицы. В инспекторе атрибутов выберите для них стиль Left Detail.

Дважды щелкните на метке 'Title' и введите новый текст 'Artist' и 'Location'.

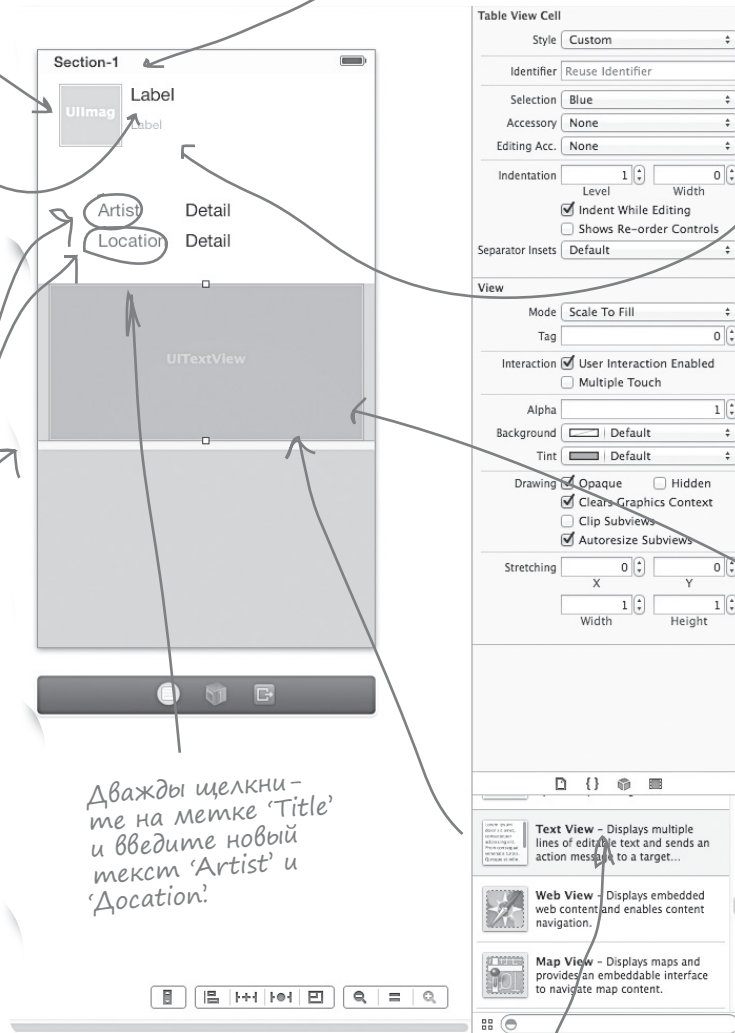
Понадобится еще одна ячейка. Перетащите ее в табличное представление.

Перетащите элемент UIImageView в левую часть ячейки. Разместите его у левого края, задав его высоту и ширину равными 60 пунктам.

Перетащите другую надпись в ячейку. Разместите ее с рекомендованным отступом у правого края UIImageView, прямо под добавленной надписью. Растяните надпись до правого рекомендованного отступа табличного представления. Выберите цвет Light Gray Color и шрифт System 12.0.

Убедитесь в том, что для ячеек выбран тип Custom. Перетащите границу ячейки, чтобы ее высота была равна 150 пунктам.

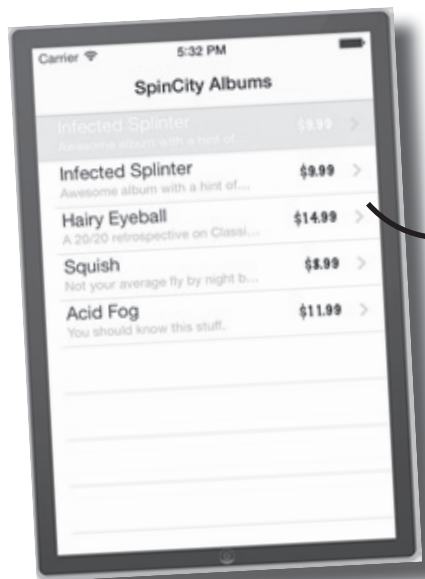
Перетащите текстовое поле в ячейку. Выверните его в ячейке с меньшим рекомендуемым верхним отступом. Выверните нижнюю, правую и левую сторону с нормальными рекомендуемыми отступами.



А теперь
запустите!



ТЕСТ-ДРАЙВ



Ничего!

Это что, шутка? Столько
работы, и ничего не появляется?
Может, мы забыли создать
какие-то связи?



**Нужно проложить путь
от главного представления
к детализированному.**

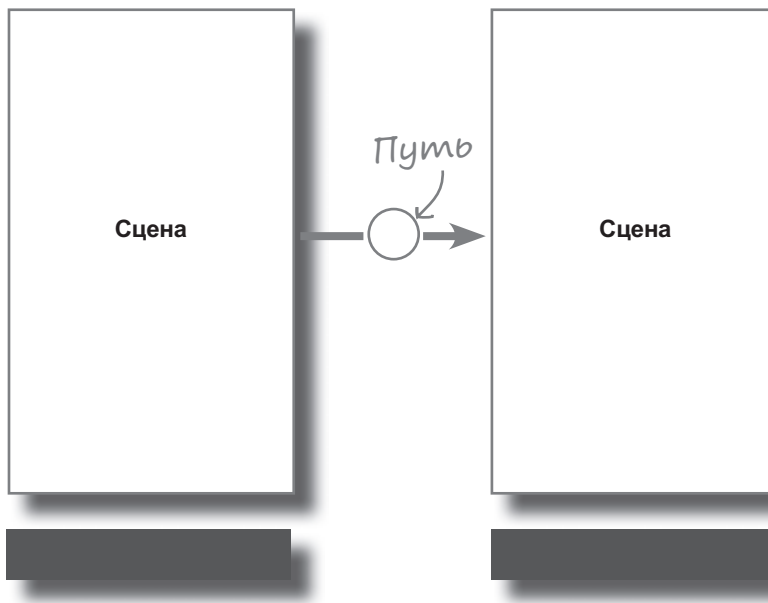
При удалении представления также
был удален ведущий к нему переход.
Раскадровка позволяет не только
редактировать представления, но и
добавлять ведущие к ним переходы.

Пути соединяют контроллеры представлений

← Которые представлены в раскладке сценами.

Пути изображают переходы между представлениями в раскладках. Они появились относительно недавно (в iOS 6) и предоставляют широкие возможности для настройки переходов. Ранее мы говорили о переходах между представлениями, но в редакторе раскладки также существует концепция **сцены**. На iPhone сцена всегда соответствует представлению, но на iPad сцен в представлении может быть несколько (вообразите представление, в котором используются две панели, постоянная и переменная, как, например, в стандартном приложении Почта). Путь описывает способ перехода от сцены к сцене, а не от представления к представлению.

Существует несколько стандартных путей (один из которых будет использован в приложении SpinCity), но вы также можете писать собственные реализации. iOS создает пути за вас, когда в приложении активизируется соответствующий переход. Сначала создается конечная сцена, затем объект пути; исходное представление вызывает `prepareForSegue:sender:`. Наконец, вызывается метод `perform` объекта пути, и переход на этом завершается.



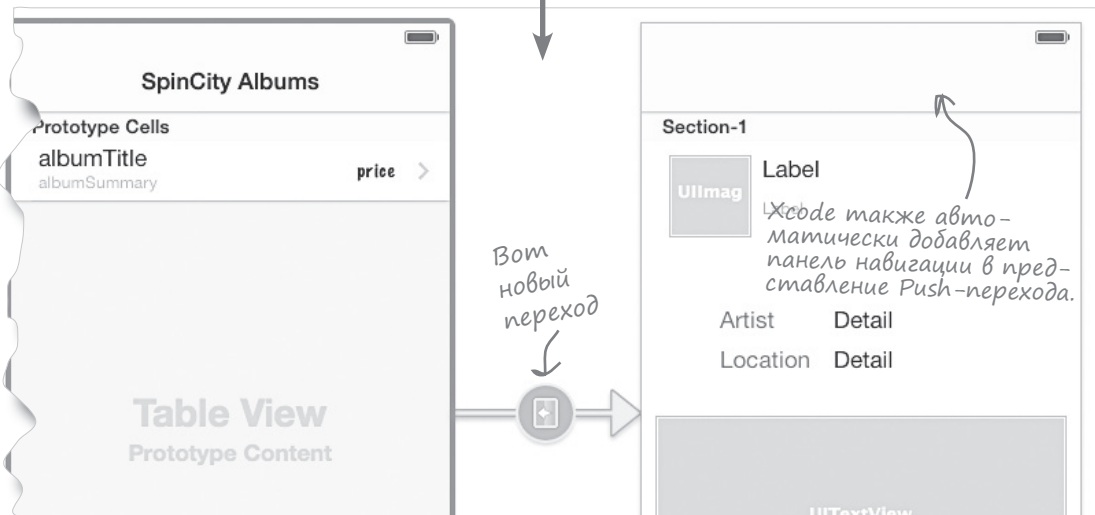
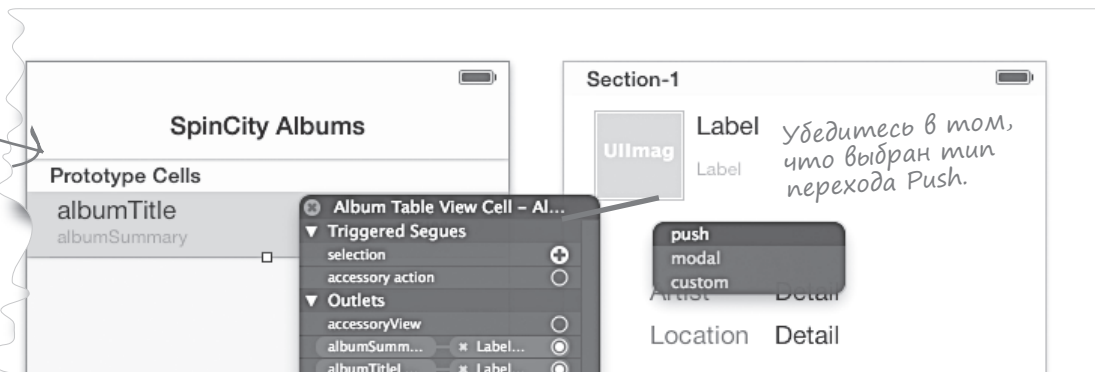
Путь определяет переход от сцены к сцене (не обязательно от представления к представлению).

А теперь за работу...

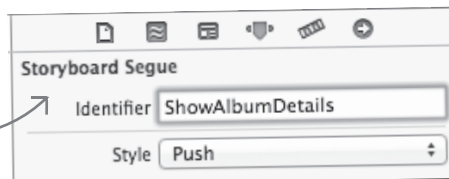
Свяжите сцены в раскадровке

В приложении SpinCity две сцены не связаны между собой, так что мы не можем добраться до только что созданного детализированного представления. Нам придется поработать в редакторе раскадровки и добавить новый переход.

Сделайте правый щелчок на ячейке табличного представления, чтобы вызвать панель путей и ссылок. Перетащите указатель мыши с нажатой клавишей *Ctrl* на второе представление.



Выделите путь и измените идентификатор на *ShowAlbumDetails*.



Часть Задаваемые Вопросы

В: Для чего нужны сцены? Кажется, что-то лишнее...

О: Потому что управление переходами должно осуществляться именно на этом уровне. Если вы хотите управлять тем, происходит ли перемещение от одного представления к другому, то теперь это стало возможно! Переходы между представлениями являются настолько важной частью большинства приложений, что их поддержка была встроена в инфраструктуру.

В: Какие еще функции выполняют пути, кроме связывания сцен?

О: Паттерн пути инкапсулирует немалый объем шаблонного кода в приложениях. Путь предоставляет удобный механизм передачи данных следующему представлению, точки подключения для выполнения завершающих действий перед закрытием представления, а также удобное место для настройки визуального сопровождения сменяемых представлений.

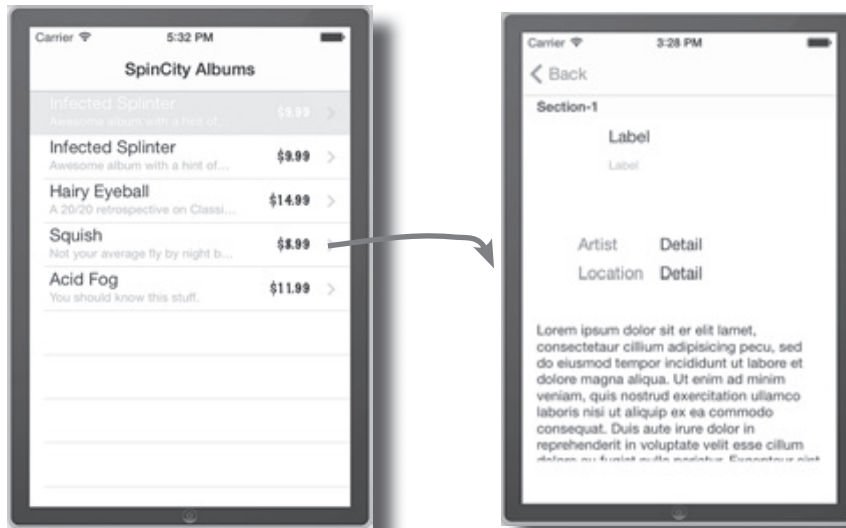
В: Почему включение пути привело к добавлению панели навигации в верхней части детализированного представления?

О: Мы выбрали Push-переход, который будет реализован через контроллер навигации. Xcode автоматически резервирует место, необходимое для панели навигации.



ТЕСТ-ДРАЙВ

Запустите!



Открывается, но...



Все это, конечно,
хорошо, но где инфор-
мация об альбоме?

**Нужно лишь заполнить представление,
как это делалось ранее!**

Детализированное представление отображается на экране, но мы не добавили действия и ссылки IBOutlet и не связали поля с данными. Необходимо передать данные представлению — по аналогии с тем, как это делалось в приложении Марко.



Упражнение

Вы уже знаете, что делать; запустите Xcode и создайте необходимые связи для детализированного представления.

1

Используйте режим Assistant Editor.

Убедитесь в том, что в раскладовке виден контроллер детализированного представления, а на панели Assistant Editor открыт файл *DetailViewController.h*.

2

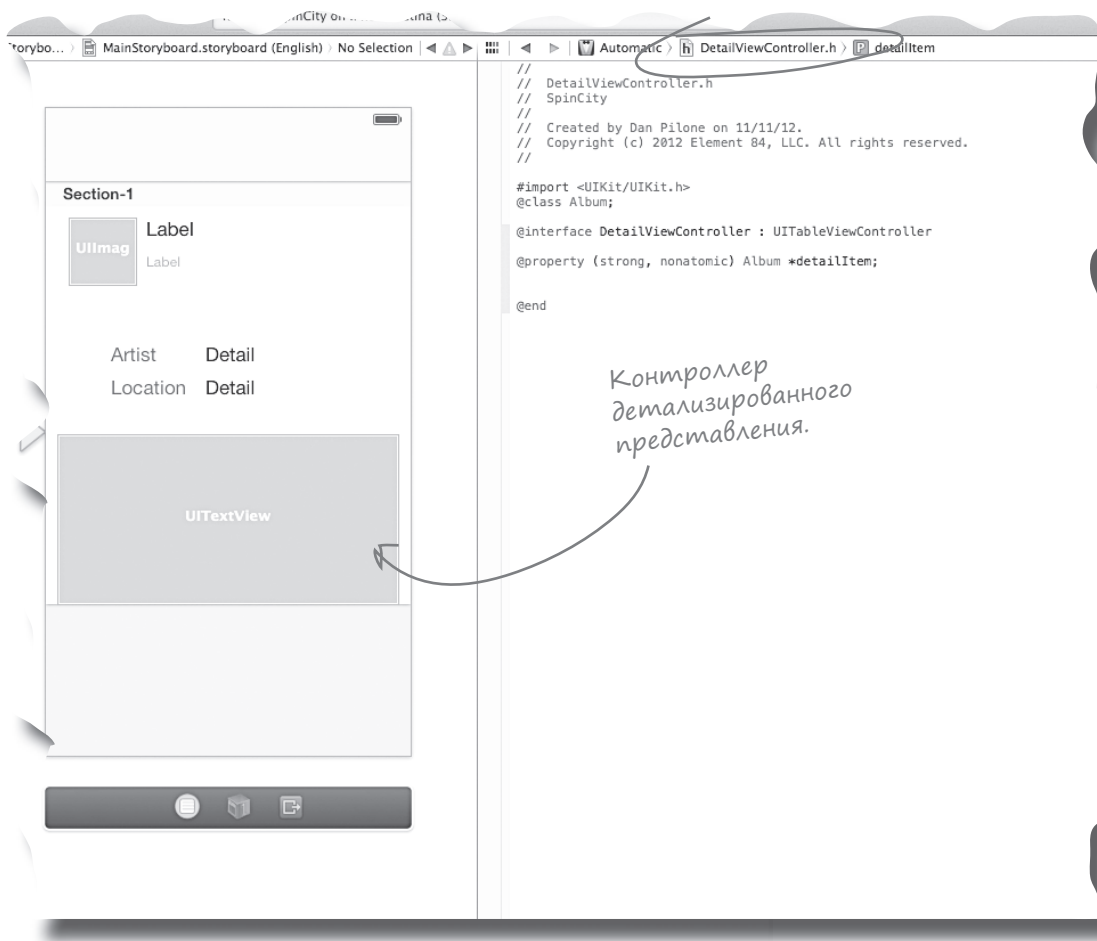
Создайте ссылки IBOutlet в детализированном представлении.

Нам понадобятся ссылки IBOutlet для надписей в представлении: `albumtitleLabel`, `priceLabel`, `artistLabel`, `locationLabel` и `descriptionTextView`.



Посмотрим, как вам следовало настроить представления.

- 1 **Используйте режим Assistant Editor.**
Убедитесь в том, что в раскладовке отображается контроллер детализированного представления, а на панели Assistant Editor открыт файл *DetailViewController.h*.

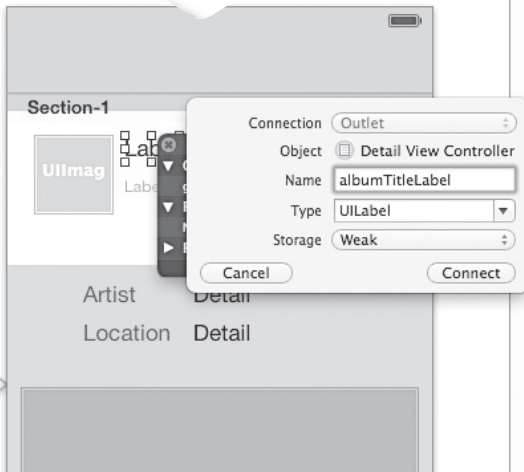


2

Создайте ссылки IBOutlet в детализированном представлении.

Нам понадобятся ссылки IBOutlet для надписей в представлении: albumtitleLabel, priceLabel, artistLabel, locationLabel и descriptionTextView.

Проследите за тем, чтобы была выделена нужная надпись!



```
// DetailViewController.h
//
// Created by Dan Pilone on 11/11/12.
// Copyright (c) 2012 Element 84, LLC. All rights reserved.
```

```
#import <UIKit/UIKit.h>
@class Album;

@interface DetailViewController : UITableViewController

@property (strong, nonatomic) Album *detailItem;

@end
```

Удерживая клавишу Ctrl, перетаскивайте указатель мыши с названия альбома в файл DetailViewController.h, чтобы создать ссылку IBOutlet с именем albumtitleLabel.

```
#import <UIKit/UIKit.h>
@class Album;

@interface DetailViewController : UITableViewController

@property (strong, nonatomic) Album *detailItem;
@property (weak, nonatomic) IBOutlet UILabel *albumtitleLabel;
@property (weak, nonatomic) IBOutlet UILabel *priceLabel;
@property (weak, nonatomic) IBOutlet UILabel *artistLabel;
@property (weak, nonatomic) IBOutlet UILabel *locationLabel;
@property (weak, nonatomic) IBOutlet UITextView *descriptionTextView;

@end
```

Итоговый код со всеми ссылками IBOutlet.



DetailViewController.h

Ссылки размещаются в заголовочном файле, но не в файле реализации...



Развлечения с магнитами

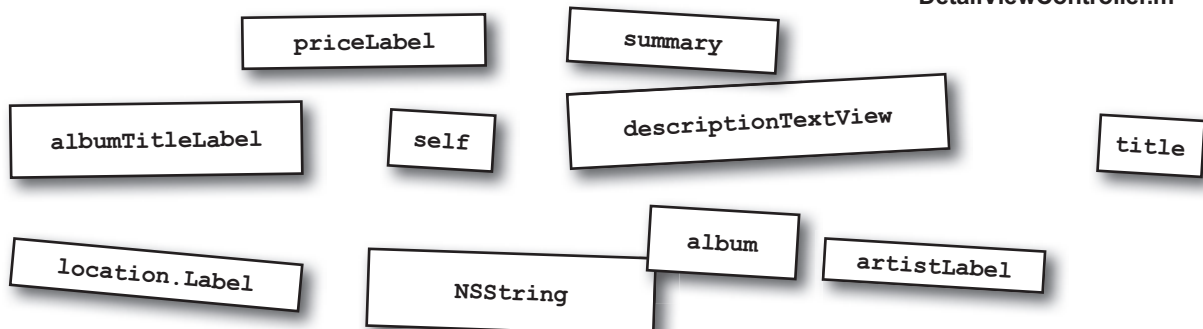
Ниже приведен код обращения к свойствам альбома в файле *DetailViewController.m*.

```
- (void)configureView
{
    // Обновление пользовательского интерфейса
    // для элемента детализированного представления.

    if (self.detailItem) {
        self._____.text = self.detailItem.title;
        self._____.text = [NSString
stringWithFormat:@"%01.2f", self.detailItem.price];
        self._____.text = self.detailItem.artist;
        self._____.text = self.detailItem.
locationInStore;
        self._____.text = self.detailItem.summary;
    }
}
```



DetailViewController.m



Часто задаваемые вопросы

В: Почему в классе `Album` для поля с кратким описанием используется имя `summary`? Почему бы не назвать его `description`?

О: Имя «description» уже присвоено методу класса `NSObject`, который должен возвращать текстовое описание экземпляра `NSObject`. По умолчанию метод возвращает имя класса и адрес объекта в шестнадцатеричной записи. Метод можно переопределить, чтобы он возвращал другую информацию — осмысленное описание объекта, которое вовсе не обязано совпадать с кратким описанием, обычно приводимым на обратной стороне компакт-диска. Мы не хотели создавать конфликт имен и поэтому использовали название «summary».



НАПРЯГИ МОЗГИ

Почему этот код еще не работает? Представление еще не готово к заполнению. Есть какие-нибудь мысли на этот счет? Подсказка: мы уже говорили об этом в текущей главе...



Развлечения с магнитами

Заполните пропуски, необходимые для связывания данных с детализированным представлением.

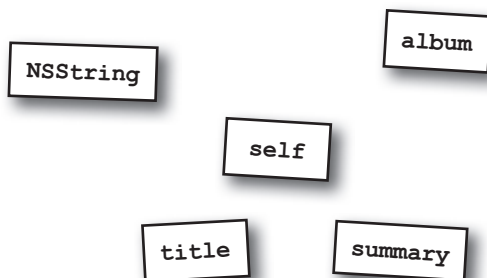
```
#import "Album.h"
```

```
- (void)configureView
{
    // Обновление пользовательского интерфейса
    // для элемента детализированного представления.

    if (self.detailItem) {
        self. albumtitleLabel .text = self.detailItem.title;
        self. priceLabel .text = [NSString
stringWithFormat:@"%$.2f", self.detailItem.price];
        self. artistLabel .text = self.detailItem.artist;
        self. locationLabel .text = self.detailItem.locationInStore;
        self. descriptionTextView .text = self.detailItem.summary;
    }
}
```



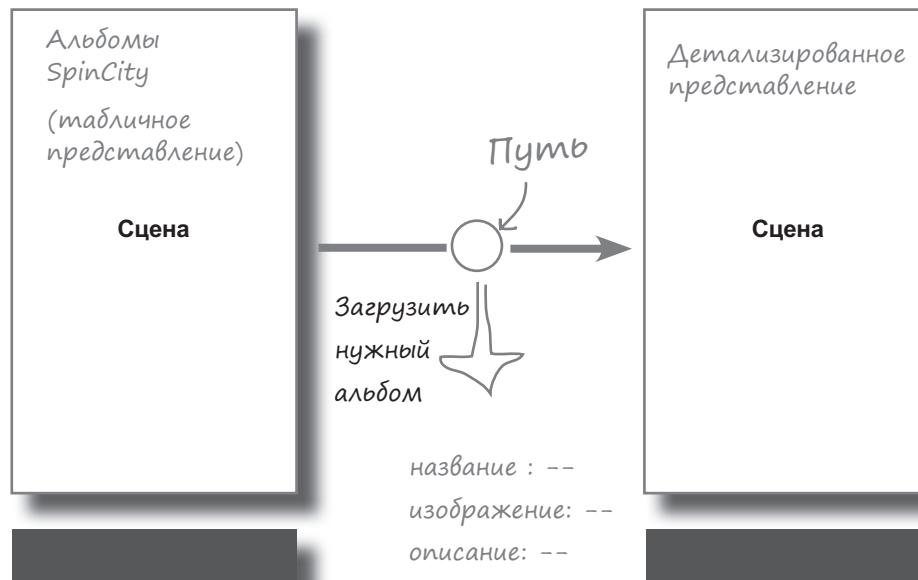
DetailViewController.m



И снова недостающим звеном является путь!

Пути позволяют подготовиться к отображению новой сцены

Пути не ограничиваются описанием перехода между сценами; в них также включаются данные, которые должны передаваться между сценами. В нашем случае детализированному представлению должна передаваться информация, которая находилась в выбранной пользователем ячейке таблицы. Благодаря этой информации детализированное представление может обратиться к нужной записи и отобразить нужные данные.



Погодите — детализированное представление нормально работало, и мы не тратили время на возню с кодом пути. Почему это нужно делать сейчас?



Для путей существует обработка по умолчанию.

Стандартный способ активизации путей использует контроллер представлений и настраивается на уровне раскадровки. Впрочем, при активизации путей на программном уровне открываются дополнительные возможности.

Обновите метод обратного вызова `prepareForSegue`

`prepareForSegue` и `performSegueWithIdentifier` — два метода, используемых iOS для работы с путями на программном уровне. Хотя контроллер представления обрабатывает путь нормально, иногда в приложении встречаются передачи управления, нуждающиеся в уточнении (например, если к представлению ведут сразу несколько переходов). И если вы создали замечательный нестандартный переход, то вы также сможете использовать его.

Метод `performSegueWithIdentifier` позволяет активизировать любой путь на программном уровне; при этом указывается строковый идентификатор, настроенный в раскадровке. Метод `prepareForSegue` уже присутствует в нашем коде; по умолчанию он переопределяется контроллером представления во время выполнения, но в нашем случае необходимо указать задействованные контроллеры представления и необходимые данные.

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)
sender
{
    if ([[segue identifier] isEqualToString:@"showAlbumDetails"])
    {
        NSIndexPath *indexPath = [self.tableView
indexPathForSelectedRow];
        Album *album = [self.albumDataController
albumAtIndex:indexPath.row];
        [[segue destinationViewController] setDetailItem:album];
    }
}
```

Единственное место, в котором нужно внести изменения.

В этом методе можно определить, какой путь и когда должен использоваться.

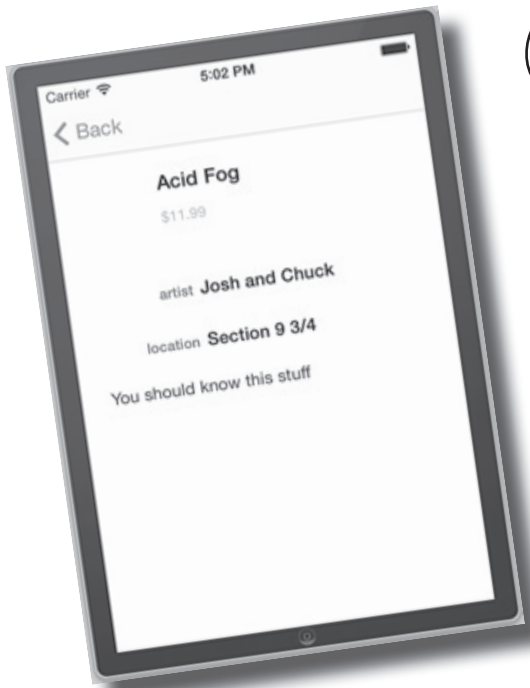


MasterViewController.m



ТЕСТ-ДРАЙВ

К этому моменту ваш код должен компилироваться и нормально работать. Запустите приложение. Если выделить ячейку в главном представлении...



Обожаю
эту сцену!

Уловили?



КЛЮЧЕВЫЕ МОМЕНТЫ



- Пути предоставляют способ управления переходами между сценами.
- По умолчанию обработка путей осуществляется в редакторе раскадровки.
- Пути можно настраивать и активизировать вручную в коде.
- На iPhone одна сцена соответствует одному представлению. На iPad одно представление может содержать несколько сцен.

Я тут подумала... А что произойдет, когда мне понадобится новая запись? Как изменить информацию в таблице?

Помощница
Роба

Сейчас добавлять новые записи можно только одним способом — прямо в коде.

Чтобы у приложения были тестовые данные, мы просто ввели их непосредственно в коде. Теперь нужно придумать более удобный способ добавления записей. Для данных, как и положено, определен отдельный класс, но сейчас пришло время преобразовать данные в более удобную форму. Таковую, как список plist!



```
- (void)initializeDefaultAlbums {
    [self addAlbumWithTitle:@"Infected Splinter" artist:@"Boppin'
Beavers" summary:@"Awesome album with a hint of Oak." price:9.99f
locationInStore:@"Section F"];

    [self addAlbumWithTitle:@"Hairy Eyeball" artist:@"Cyclops"
summary:@"A 20/20 retrospective on Classic Rock." price:14.99f
locationInStore:@"Discount Rack"];

    [self addAlbumWithTitle:@"Squish" artist:@"the Bugz"
summary:@"Not your average fly by night band." price:8.99f
locationInStore:@"Section A"];

    [self addAlbumWithTitle:@"Acid Fog" artist:@"Josh and
Chuck" summary:@"You should know this stuff." price:11.99f
locationInStore:@"Section 9 3/4"];
}
```

Помните? Все данные сейчас жестко зафиксированы в программном коде...

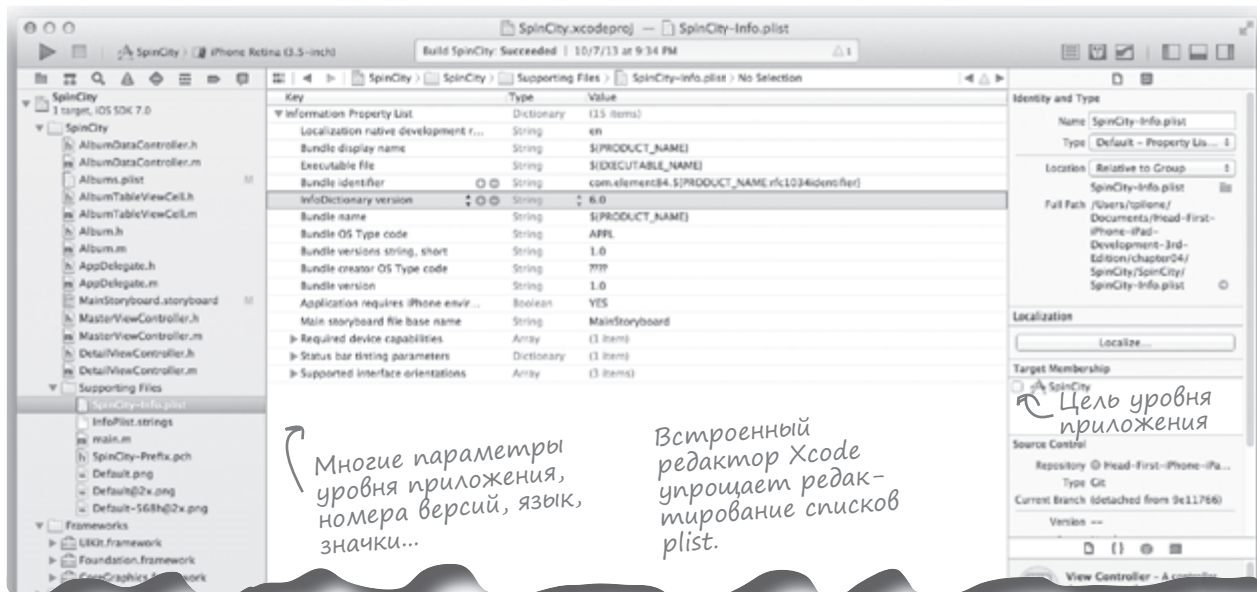


AlbumDataController.m

список

Для этого есть приложение

Сокращение **Plist** происходит от слов «property list» («список свойств»); эта форма представления данных уже довольно давно существует в OS X. Более того, в нашем приложении уже используются несколько списков plist. Мы уже работали с самым важным списком plist — файлом *Info.plist* вашего приложения. Он строится средой Xcode при создании проекта, и кроме значков приложения в нем хранятся такие сведения, как главный файл раскадровки, загружаемый при запуске приложения, версия приложения и т. д. Xcode может создавать и редактировать такие списки plist, как и любые другие файлы. Чтобы ознакомиться с содержимым списка, щелкните на файле *SpinCity-Info.plist*.



Часто задаваемые вопросы

В: Откуда взялись списки plist?

О: Списки plist характерны для программирования Mac и iOS, хотя сама концепция восходит к инфраструктурам NeXTSTEP и GNUstep. Списки могут использоваться для хранения данных, относящихся к типам Core Foundation: CFString, CFNumber, CFBoolean, CFDate, CFData и CFDictionary. Редактор может преобразовать их в формат XML или в двоичный формат.

В: Мы будем работать с XML в своем приложении?

О: Нет, мы воспользуемся инфраструктурами iOS для загрузки списка plist (а также его разбора и т. д.). После загрузки данных мы получим массив словарей, очень удобный для работы с данными альбомов.

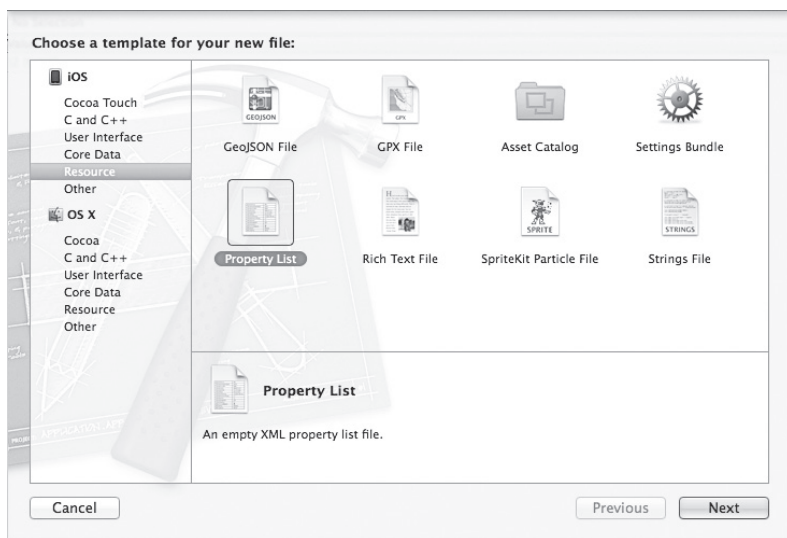
Создайте новый список свойств

Все встроенные типы, которые использовались нами ранее (такие, как NSArray и NSString), могут загружаться или сохраняться в списках plist автоматически через протокол NSCoding. Мы можем воспользоваться этим обстоятельством и вынести тестовый список альбомов из исходного кода в список plist.

1 Создайте пустой список plist.

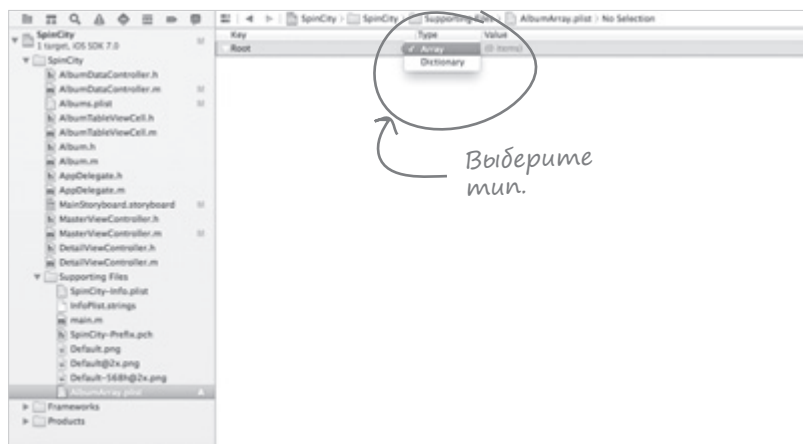
Вернитесь в Xcode и откройте папку **Supporting Files**. Сделайте правый щелчок на папке **Supporting Files**, выберите команду **New file**→**Mac OS X Resource** и **Property List**. Присвойте новому списку имя *AlbumArray.plist*.

Списки plist используются в программировании не только для iOS, но и для Mac, однако включены они в эту категорию.



2 Преобразуйте список plist в массив.

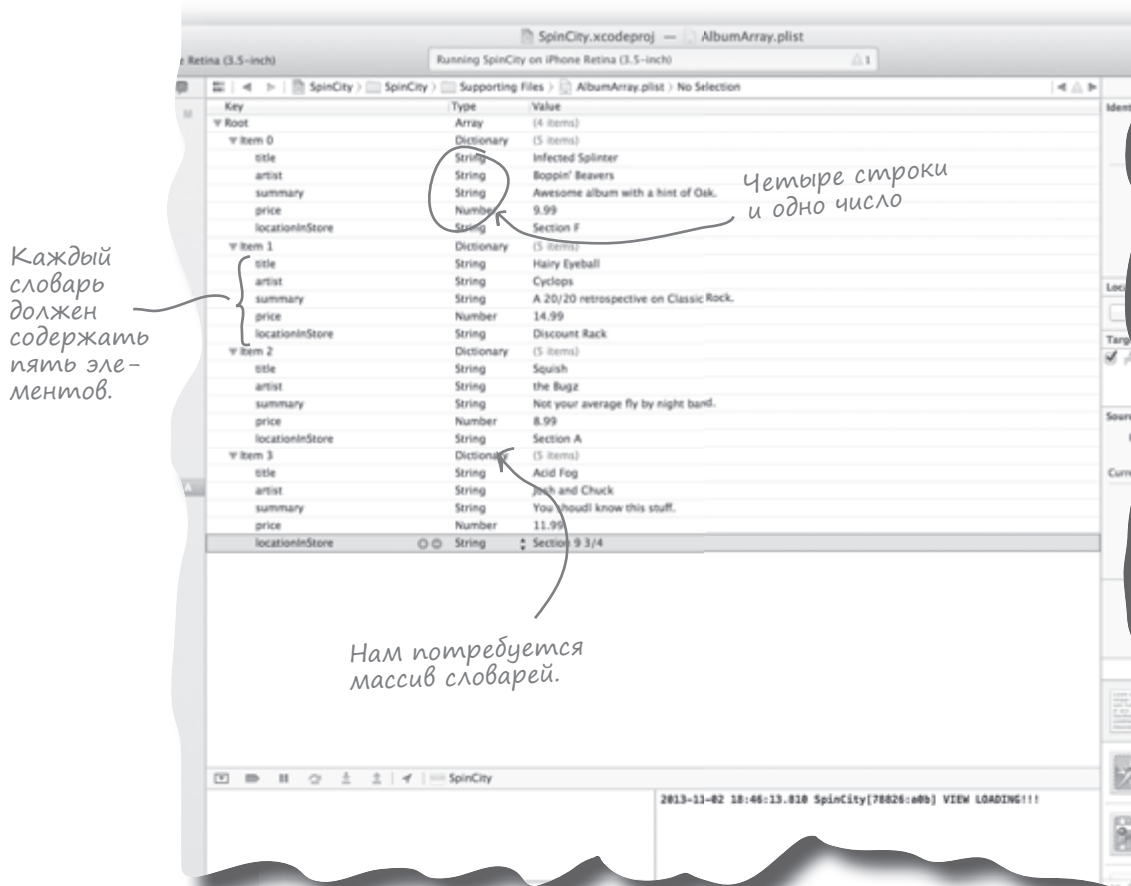
Вернитесь в Xcode и откройте новый файл *AlbumArray.plist*. В редакторе выберите в поле **Тип** массив (Array).



3

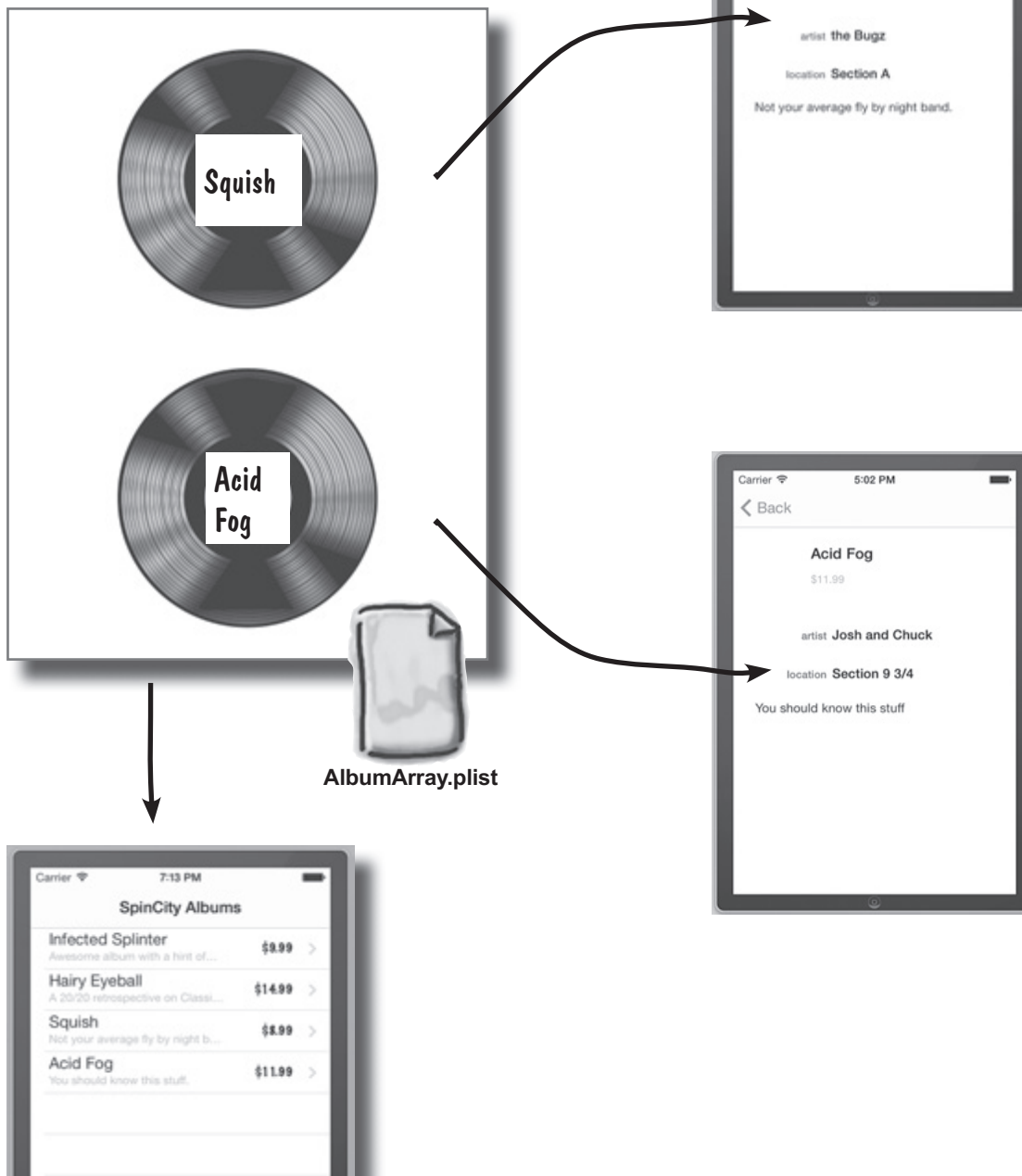
Поместите данные в список plist.

Так как мы собираемся извлечь из списка plist полную информацию об альбомах, описания альбомов из кода следует переместить в файл *AlbumArray.plist*.



Мы должны загрузить каждый альбом из списка

При реализации кода `AlbumDataController` метод `initializeDefaultAlbums` использовался для генерирования массива с именем `_albumList`. Теперь мы переключимся на создание массива на базе списка *plist*. Это будет массив словарей...



Преобразование данных в список plist одной простой операцией

Так как инициализация данных выполняется методом `initializeDefaultAlbums` из файла *AlbumDetailController.m*, достаточно привести этот метод к следующему виду:

```
- (void)initializeDefaultAlbums {
    NSString *pathToAlbumsPlist = [[NSBundle mainBundle]
    pathForResource:@"AlbumArray" ofType:@"plist"];
    NSArray *defaultAlbumPlist = [NSArray arrayWithContentsOfFile:
    pathToAlbumsPlist];
    for (NSDictionary *albumInfo in defaultAlbumPlist) {
        [self addAlbumWithTitle:albumInfo[@"title"]
        artist:albumInfo[@"artist"] summary:albumInfo[@"summary"]
        price:[albumInfo[@"price"] floatValue] locationInStore:albumInfo[@"locationInStore"]];
    }
}
```

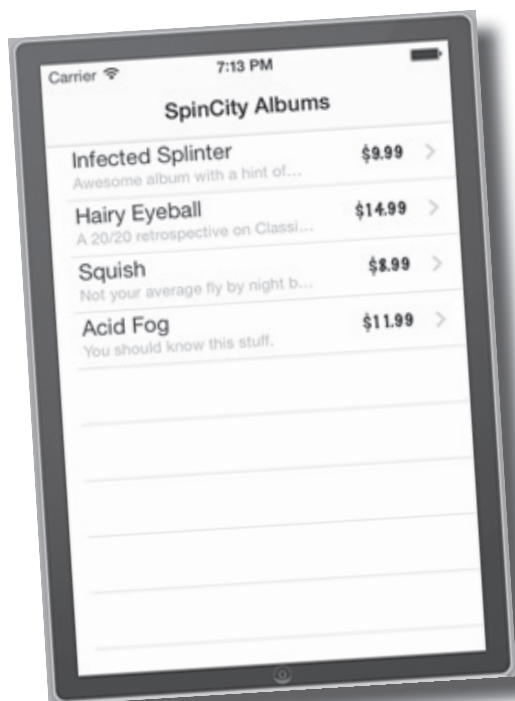


AlbumDataController.m



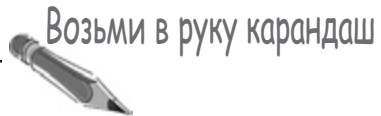
ТЕСТ-ДРАЙВ

Запустите приложение и проверьте, как оно работает...



Роб в восторге!





Какие из следующих утверждений о списках plist истинны?
И почему их стоит использовать в программах?

☐

Это простой способ организации хранения данных, хорошо подходящий в начале работы над программой.

☐

Списки plist не поддерживают более сложные типы данных — только строки.

☐

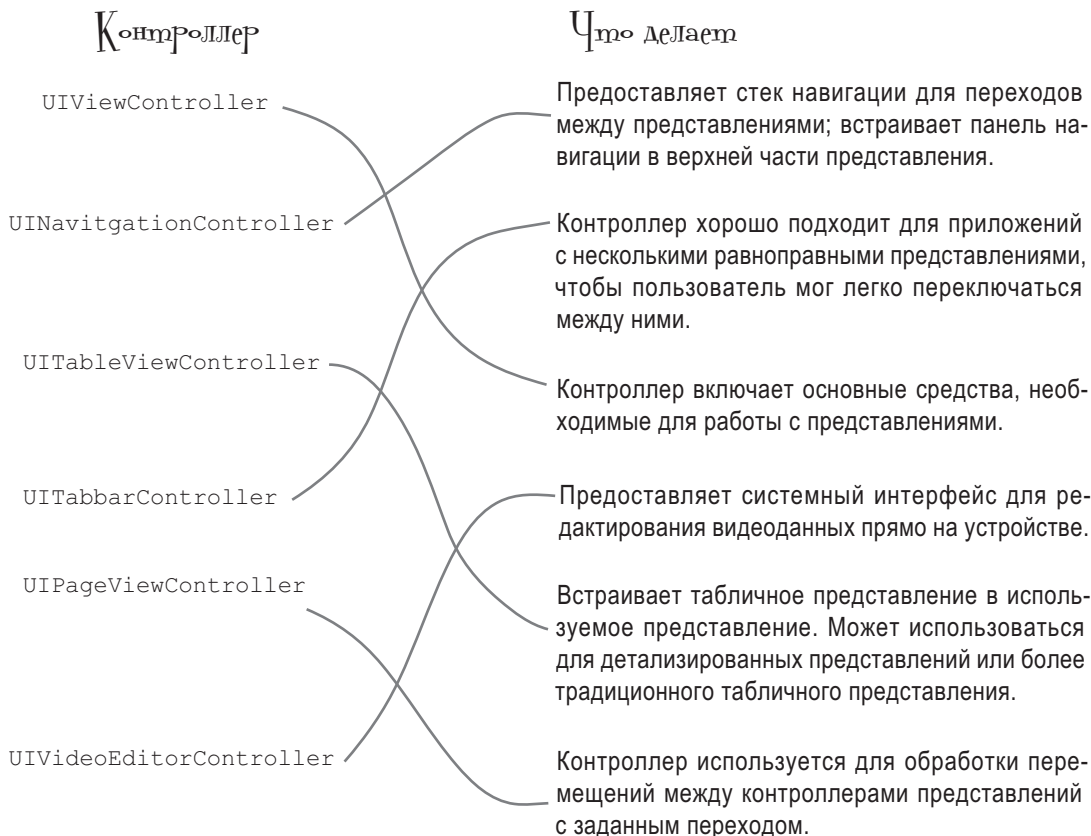
Это структура базы данных.

☐

Списки plist хранятся в формате XML.

КТО И ЧТО ДЕЛАЕТ? РЕШЕНИЕ

Соедините каждый контроллер с описанием того, что он делает.



Возьми в руку карандаш Решение

Какие из следующих утверждений о списках plist истинны?
И почему их стоит использовать в программах?

- | | |
|---|--|
| <input checked="" type="checkbox"/> Это простой способ организации хранения данных, хорошо подходящий в начале работы над программой. | <input type="checkbox"/> Списки plist не поддерживают более сложные типы данных — только строки. |
| <input type="checkbox"/> Это структура базы данных. | <input checked="" type="checkbox"/> Списки plist хранятся в формате XML. |

Списки plist представляют собой файлы XML. Позднее мы поговорим о базах данных, но самым простым вариантом является SQLite.



Ваш инструментарий представлений

Глава 4 осталась позади, а ваш инструментарий пополнился приложениями с несколькими представлениями.

Табличные представления

- ☐ Поддерживают настройку, при которой перестают походить на традиционные таблицы.
- ☐ Чтобы преобразовать шаблонное детализированное представление в табличное представление, его придется отредактировать.
- ☐ Динамические ячейки табличных представлений используются в том случае, если ячейки должны изменяться в зависимости от данных.
- ☐ Статические ячейки табличных представлений используются в том случае, если макет ячеек должен оставаться постоянным.

Пути

- ☐ Представляют переходы между сценами.
- ☐ В приложениях iPad одно представление обычно состоит из двух сцен.
- ☐ Могут настраиваться в программном коде.

Списки plist

- ☐ Хорошо подходят для представления массивов и словарей чисел и строк.
- ☐ Поддерживают хранение только строк и числовых данных.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Пути предоставляют способ управления переходами между сценами.
- По умолчанию обработка путей осуществляется в редакторе расстановки.
- Пути можно настраивать и активизировать вручную в коде.
- На iPhone одна сцена соответствует одному представлению. На iPad одно представление может содержать несколько сцен.

5 Процесс рецензирования, дизайн и устройства

Как жить в «яблочном» мире

Мама говорит, что если
я не буду есть по яблоку
каждый день, от меня отвер-
нутся все друзья и знакомые.
Надо почистить и нарезать
дольками!



Разработка iOS не ограничивается одним программированием.

Наверняка вы уже слышали всевозможные ужасы. Процесс рецензирования Apple известен своей строгостью и обилием правил, которые должны соблюдать программисты. Да, не ждите, что все пойдет как по маслу, но если вы знаете, что делаете, — все отнюдь не так плохо. А кроме того, когда ваше приложение будет одобрено, перед вами откроется невероятно популярный магазин App Store... с множеством покупателей, готовых расстаться со своими долларами. Разве не заманчиво?



Запустить Xcode. Загрузить проект из Git. Перетащить несколько элементов, написать немного кода Objective-C... Звучит так просто. Нет, правда, слишком просто...

Да, написать приложение несложно — когда вы знаете, что делаете. Но чтобы ваше приложение продавалось в App Store, одним программированием дело не ограничится.

Магазин App Store — это мир Apple. Эта компания устанавливает правила игры, и очень серьезно относится к своим обязанностям владельца магазина (а также производителя ваших устройств и программного обеспечения).

Если вы начнете заниматься разработкой приложений iOS, до вас неминуемо начнут долетать слухи. Некое замечательное приложение было трижды или четырежды отвергнуто в процессе рецензирования, а тем временем у приложений Facebook и Apple Trailers вышло четыре новые версии. Вроде бы сущая мелочь — URL в описании, плохо выбранное имя — и приложение отклоняется.

Да, ваше приложение должно быть одобрено рецензентами Apple. И оно действительно может быть отклонено по многим причинам. Но... отказы чаще всего связаны с мелочами, которые легко исправить. И если вы знаете, что делаете, вам удастся легко избежать долгого ожидания и дорогостоящей переработки.

Это мир Apple... просто вы в нем живете

Между разработкой для iOS и разработкой для Android (а также для большинства других платформ) существует принципиальное отличие: App Store — жестко управляемое сообщество. Это означает, что Apple имеет **исключительное право** принимать или отклонять приложения в своем магазине на основании определенных критериев. Тем самым гарантируется единство стиля и некоторая степень качества. У Apple действует система цензуры, так что непристойные или излишне жестокие приложения не принимаются.

Итак, если вы хотите работать в пространстве iOS, можете **считать этот процесс частью своей жизни**. В процессе разработки могут происходить события, вам неподконтрольные, а иногда плохо прогнозируемые — например, выход новой версии iOS и ее последствия для ваших приложений. Скажем, iOS 7 внесла ряд принципиальных изменений в дизайн приложений, и внезапно «новый пользовательский интерфейс» оказался на первых позициях в списке текущих задач разработчиков. Процедура рецензирования продолжается от недели до месяца, а отказ обычно означает повторную отправку и полное прохождение всей процедуры.

Вам также придется изучить ряд документов, которые выходят за рамки написания кода приложений. Прежде все это...

App Store... Если вы понимаете, как у Apple организована продажа приложений, это многое упрощает.

Нравится вам это или нет — правила определяет Apple. Как бы вы к этому ни относились — как к стремлению к созданию лучшего, более безопасного устройства, или же к попытке тотального контроля, — в мире iOS придется жить по правилам Apple.



Human Interface Guidelines (HIG). Apple хочет, чтобы ваши приложения работали определенным образом... И эти правила лучше не нарушать.

Рекомендации по программированию для iOS — изложение «пути Apple», который также должен стать «вашим путем».





Отказы Apple

под увеличительным стеклом

Мы не приводим текст реальных отказов — Apple не разрешает это делать. Тем не менее такие ситуации типичны, а отказы по таким причинам встречаются сплошь и рядом.

Интересно узнать типичные причины отказа? Вот несколько примеров. Прочитайте внимательно! В большинстве случаев вы быстро поймете, как избежать такой неприятной ситуации.

To: Разработчик Дин

From: Рецензенты Apple

Subject: Отказ по приложению

Мы проверили ваше приложение и определили, что в нем нарушаются некоторые рекомендации.

«Приложения со ссылками на внешние механизмы приобретения контента или подписки — например, кнопка “Купить”, ведущая на веб-сайт для приобретения электронной книги, — будут отклоняться».

Apple крайне жестко контролирует ссылки на внешние сайты в приложениях. Если эти сайты продают какой-либо цифровой контент: книги, музыку, фильмы, что угодно — ваше приложение не пройдет рецензирование.

И не думайте, что у рецензентов Apple не хватит времени на проверку каждой ссылки. Они фанатично следят за тем, чтобы покупка цифрового контента в приложениях осуществлялась только через механизм In App Purchase магазина App Store.

В наши дни многие приложения работают как на iPhone, так и на iPad. Если это относится к вашему приложению, то для прохождения рецензирования оно должно содержать значки для разрешений экранов разных устройств, а также предоставлять графику, отображаемую в iTunes и App Store.

К счастью, рекомендации iOS Human Interface Guidelines (см. далее) предельно четко объясняют, какие ресурсы должны включаться в приложение. Прочитайте рекомендации и приложите графику для нужных разрешений. Все просто!

To: Компания «И так сойдет»

From: Рецензенты Apple

Subject: Отказ по приложению

«Мы проверили ваше приложение и определили, что в нем не хватает значков для других разрешений экрана».



Отказы Apple под увеличительным стеклом

To: Компания «Долой старье»

From: Рецензенты Apple

Subject: Отказ по приложению

«Мы проверили ваше приложение и определили, что оно не проходит проверку работоспособности. Ваше приложение не работает в iOS 4.0».

Старые телефоны уже не актуальны, верно? Нет, неверно! Apple позволяет указать, какие версии iOS вы поддерживаете, но если какая-то версия заявлена — Apple проследит за тем, что она **ДЕЙСТВИТЕЛЬНО** поддерживается.

Что это означает для вас? Тестирование на всех устройствах, на которых должно работать ваше приложение... Без исключений!

Даже на телефонах и iPad с гигантским объемом памяти 64 Гбайт компания Apple серьезно следит за тем, чтобы ваше приложение вело себя разумно и не слишком эгоистично. Хотя писаных правил на этот счет нет, старайтесь использовать только ту память, которая вам действительно необходима.

Контент, сгенерированный пользователем, обычно приемлем, но если ваше приложение загружает большой объем информации при запуске (особенно без участия пользователя) — берегитесь. Возможно, вас ждет отказ...

To: Компания «Передовые разработки»

From: Рецензенты Apple

Subject: Отказ по приложению

«Мы проверили ваше приложение и определили, что оно загружает слишком большой объем контента. Приложения ограничиваются по объему ресурсов, которые могут потребляться при исходной загрузке».



Упражнение

Мы привели лишь несколько примеров, но возможно, вы уже представляете себе основные причины, по которым Apple может отклонить ваше приложение. На основании того, что вы узнали, попробуйте определить, почему приведенное ниже приложение почти наверняка будет отклонено в процессе рецензирования.

А если уж вы совсем жестоки, предложите не одну, а целых три причины, по которым Apple покажет этому приложению «красную карточку».



Часть Задаваемые Вопросы

В: Нечестно! Это тоталитаризм!!

О: Да, иногда все выглядит именно так. Но весь этот процесс — плата за то, что вы увидите свое приложение на устройстве iOS. Иного пути просто нет. И конечно, награда того стоит: магазин App Store по заработкам существенно опережает своих конкурентов и поэтому является лучшим местом для продажи приложений!

В: Могу ли я определить, не будет ли мое приложение отклонено, до отправки на рецензирование?

О: Нет, не можете — по крайней мере не на 100%. Лучшее, что вы можете сделать, — тщательно ознакомиться с правилами рецензирования App Store перед тем, как браться за программирование. (Да, именно перед тем!) Если в приложении присутствует какая-либо неоднозначная функциональность, попробуйте поискать в App Store другие приложения с аналогичными потенциальными проблемами. Если такие приложения прошли рецензирование, вероятно, их создатели как-то справились с вашими трудностями.

И все же ничего нельзя сказать с уверенностью до того, как ваше приложение отправится на рецензирование. На практике это означает, что вы должны включить в график как минимум один отказ и повторную отправку! Если возникнут какие-то непредвиденные обстоятельства, по крайней мере бизнес-график не будет безнадежно испорчен!

В: Они отклонили мое приложение! Как мне его отправить повторно?

О: Это зависит от причины отказа. Если из-за мелких проблем в метаданных — иногда удастся вернуться в начало очереди; это лучший вариант.

К сожалению, большинство отказов сопряжено с изменениями в коде, выходящими за рамки метаданных. Это означает, что вам придется заново отправить все приложение... И пройти весь процесс рецензирования заново.

В: Можно ли опротестовать отказ?

О: Конечно! Когда ваше приложение отклоняется, всегда можно подать апелляцию. Однако рассмотрение апелляций занимает много времени, так что подавайте их, только если вы **действительно** уверены в своей правоте. В противном случае все кончится потерями времени на апелляцию и на повторную отправку.

В: Как все это можно планировать? Полный хаос...

О: Да, строить графики в такой ситуации непросто. Но с этим все равно ничего не поделаешь. Опытные разработчики часто отводят на процесс отправки (и повторной отправки) не менее месяца.

В: Но некоторые требования просто нелепы! Я не могу использовать ссылки на внешние сайты? А на свой собственный сайт?

О: Вообще-то запрещены не все внешние ссылки... но многие. Проблема с внешними ссылками заключается в том, что если существует путь навигации с внешнего сайта на какую-нибудь платежную систему, Apple, скорее всего, отклонит ваше приложение. Даже если вы не хотели ничего плохого, Apple рассматривает это как попытку обойти свой механизм оплаты (In App Purchase или IAP) — а к подавлению таких попыток в Apple относятся очень серьезно.

Что еще хуже, внешние ссылки могут отрицательно повлиять на рейтинг вашего приложения. Игре для детей, которая могла бы стать хитом в младших классах, вдруг присваивается взрослый рейтинг — из-за контента в трех страницах от той, на которую ведет ваша ссылка. Да, у нас такое случалось... Будьте очень, очень осторожны при использовании внешних ссылок.



Упражнение Решение

Отказ... Какое неприятное слово! Будем надеяться, что вы нашли хотя бы пару причин, по которым это приложение может быть отклонено. Обязательно разберитесь, что здесь недопустимо с точки зрения Apple. Чем лучше вы будете видеть происходящее «глазами Apple», тем реже вы будете получать сообщения с отказами.

Как вы думаете, Apple серьезно относится к своим товарным знакам? И не сомневайтесь. Даже не пытайтесь отправить вайтс. Даже не пытайтесь отправить iPhone и iPad приложение, в котором слова iPhone и iPad записаны с ошибками или с неверным регистром символов.

Заодно удалите все логотипы Apple... или все, что можно принять за логотип Apple. Если вы этого не сделаете, это тоже может стать причиной отказа.

Компании Apple не нравится, когда сторонние приложения пытаются воспользоваться ее популярностью, поэтому к таким вещам рецензенты тоже относятся серьезно.

Внешняя ссылка, в которой предлагается оформить подписку в Интернете? Будьте уверены — Apple не оставит ее без внимания.

Выглядит как в iOS 6! Если эта часть не будет работать в iOS 7 — значит, вам не повезло.



Проверка устройства — обязательная часть

Первая модель iPod вышла в 2001 году, а первая модель iPhone — в 2007-м. В техническом отношении эти устройства разделяют около 32 000 лет. И хотя Apple активно содействует обновлению ОС, обновление старых устройств с какого-то момента становится невозможным. Это означает, что вы должны решить, какие версии ОС, а следовательно, какие устройства, будет поддерживать приложение.

Еще важнее другой вопрос — что это будет означать для старых устройств, которые не могут быть обновлены до поддерживаемой вами версии iOS? Старые устройства могут поддерживать или не поддерживать камеру, запись видео, внутренние динамики, GPS... И это только начало!

Корректная обработка* подобных ситуаций получила название «**проверка устройства**» (device checking). Вы должны использовать этот процесс, чтобы гарантировать нормальное поведение всех устройств, на которых может запускаться ваше приложение — даже если они не обладают программными или аппаратными возможностями, которые могут использоваться вашим приложением.



Так как же работает проверка устройств?

Пример проверки устройства: камера

Как работает проверка устройства? Предположим, вашему приложению нужно, чтобы устройство поддерживало камеру. Но как только у приложения возникает необходимость в каком-то конкретном устройстве, приходится учитывать новые дополнительные требования.

Любое приложение, написанное для iPhone, также **должно** работать на iPod и на iPad. Это означает, что при использовании функций, присутствующих только на части поддерживаемых устройств — например, камеры, необходимо проверять их доступность... и обрабатывать ситуации, в которых нужное устройство недоступно.

Итак, возьмем камеру: класс UIImagePickerController содержит метод для проверки камеры.

```
[UIImagePickerController  
isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]
```

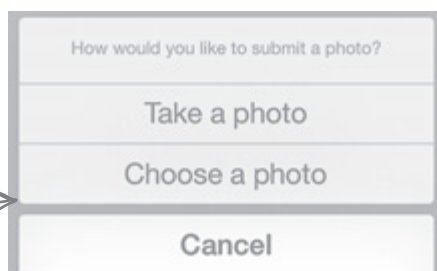
Камера является источником данных (source); приведенная команда проверяет, что источник доступен... перед тем, как его использовать.

Да, вы можете указать в описании «только для iPad», но лучше поддерживать все устройства (iPad, iPod, iTouch, iНовинка2015), использующие версию iOS, для которой предназначено ваше приложение.

iOS берет на себя технические подробности

Система iOS достаточно умна для определения функций, поддерживаемых конкретным устройством. Таким образом, запросив у UIImagePickerController типы источников данных, вы сможете решить, какие варианты следует предложить пользователю.

Так выглядит контроллер, допустим, на последней модели iPhone 5.



Обратиться к камере, сделать фотографию, а затем вернуться и поместить новое изображение в переменную. Вы передаете управление классу UIImagePickerControllerSourceTypeCamera, он делает все остальное.

Перейти к библиотеке фотографий, выбрать изображение, вернуться и работу выполняет класс UIImagePickerControllerSourceTypePhotoLibrary.

Просто вернуться к исходному представлению — вероятно, некоторой разновидности UIImageView.

Хмм... устройство поддерживается, функция недоступна

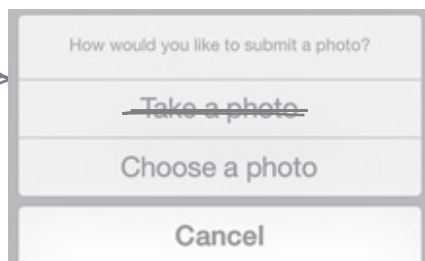
Итак, проверка таких источников данных, как камера, обязательно. Но даже если вы *проверяете* наличие источника, это не означает, что на устройстве действительно *присутствует* этот источник. Помните iPod третьего поколения?



Относительно новая модель iPod, которую, вероятно, стоит поддерживать... Но на ней нет камеры.

В подобных ситуациях следует определить, что должно происходить при отсутствии нужной функции. Вы можете полностью заблокировать соответствующую часть своего приложения. Конечно, это довольно радикальное решение — лучше поискать подмножество функций, которые могут использоваться вашим устройством.

Итак, получение новой фотографии на многих iPod и старых iPad не работает...



...Но выбор существующей фотографии вполне может поддерживаться вашим приложением — и старыми устройствами пользователей.

При работе с изображениями почти всегда существует хороший вариант: фотогалерея. Если на устройстве нет камеры, пользователь может загрузить существующие фотографии. Если большая часть функциональности приложения сохраняется, возможно, вам удастся ограничиться отключением некоторых кнопок (если возможно, постарайтесь скрыть элементы управления, которые не имеют смысла для данного устройства).

Если приложение вообще не может работать без камеры (доступа к сети, акселерометра и т. д.), укажите эти требования в описании приложения, чтобы пользователи с такими устройствами не могли даже установить у себя ваше приложение.



Значит, Apple следит за тем, чтобы мое приложение работало на их устройствах. Хорошо хоть, что они не заставляют меня использовать конкретный дизайн, чтобы мое приложение выглядело «стандартно».

Вообще-то... они следят и за этим.

Знакомьтесь: **Human Interface Guidelines**.

Вероятно, вы уже заметили, что все приложения на устройствах iOS (кроме игр) используют сходный дизайн и схемы взаимодействия с пользователем. Так создается фирменный «стиль» приложений iOS, благодаря которому пользователю становится проще освоить новое приложение. Это называется «стандартом минимальной доступности»: пользователь уверен в том, что он относительно легко освоит почти любое приложение, потому что он уже работал с другими приложениями, которые выглядят и работают похожим образом.

Все эти взаимодействия подробно описаны в документе **Human Interface Guidelines**. Это еще один документ с портала Developer Portal, описывающий процесс работы пользователя с приложением.

В iOS 7 правила оформления и поведения приложений были существенно переработаны. Новая версия HIG поможет вам перейти на новые правила iOS 7.

Вас раздражают требования к дизайну приложений? Это нормально... Многие чувствуют то же самое — до того, как хорошенько подумают.

Правила HIG помогают, а не мешают

Документ HIG полон правил, которые упрощают вашу работу. Это длинный документ, но его определенно стоит прочитать, чтобы позднее вы могли обращаться к нему время от времени, чтобы освежить память. В HIG рассматривается практически все, от рекомендаций по дизайну до использования значков. Вы узнаете, как спроектировать приложения «в стиле» стандартных приложений Mac — таких, как Почта...

Вы уже использовали панель навигации в приложениях для iPhone... А здесь используется контроллер разделенного представления iPad.

Хотите знать, какую высоту должны иметь эти панели? Загляните в HIG.

Что должно находиться на верхней панели содержания? Загляните в HIG.

Описаны даже значки, которые должны находиться на панелях, их расположение и размеры.

Включение временно-го текста в поле поиска и других текстовых полях также оговорено в HIG... с подробностями.



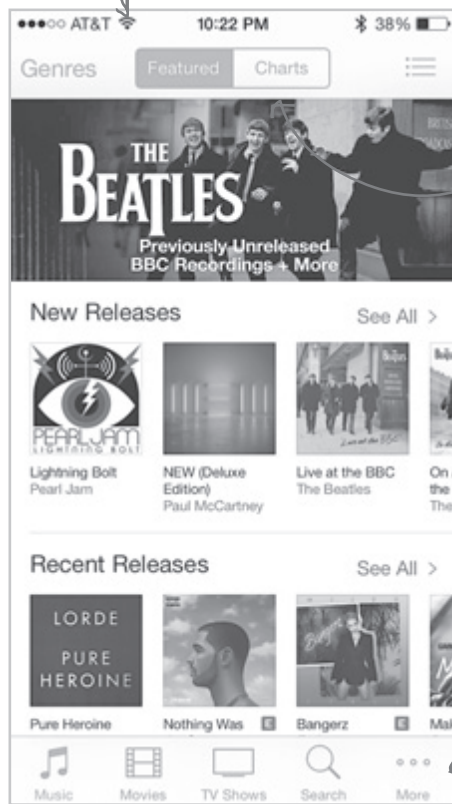
Панель детализации остается в этом месте, а табличное представление фиксируется здесь. Обе панели прокручиваются независимо друг от друга. Этот дизайн встречается во многих приложениях iOS!

Вы уже пнемногу привыкаете к правилам HIG...

А может, во время чтения вам покажется, что в HIG всего лишь документируется то, к чему вы уже привыкли... И будете правы! Каждый раз, когда вы используете такое приложение, как Почта, ваши глаза (и пальцы) привыкают работать с приложением, спроектированным по правилам HIG.

Другой пример — приложение iTunes...

Сегментированный элемент управления... HIG помогает реализовывать стиль оформления, соответствующий стилю других приложений.



В HIG указано, какие элементы управления можно разместить на верхней панели и как их использовать... Внезапно ваше приложение начинает выглядеть «знакомо» для пользователя. И это хорошо!

Контроллер панели закладок, другой стандартный элемент управления. А «стандартный» означает, что этот элемент почти всегда ведет себя одинаково... Это не тот аспект, в котором ваше приложение может блеснуть оригинальностью.

В HIG содержится много информации о том, как использовать панели вкладок, как организовать навигацию и какие значки можно использовать... Все это сильно упрощает работу дизайнера приложения.

Часть Задаваемые Вопросы

В: Понятно... Но ведь есть приложения, для которых правила HIG не нужны?

О: Вообще-то нет. Есть приложения, для которых правила HIG менее актуальны... как, например, игры, в которых вся графика создается дизайнером. Но даже в этом случае HIG помогает спроектировать естественные взаимодействия «в стиле iOS». И это очень важно...

Выходит, все ради того, чтобы не отставать от Apple и iOS 7, верно? «Новый стиль приложений iPhone»?

Нет. Правила HIG имеют отношение к проектированию приложений, но не ограничиваются визуальным дизайном.

Информационная архитектура — один из важных факторов проектирования приложений, с которыми будут работать живые люди. Версия iOS7 значительно изменила оформление и поведение iOS, но также ее выход был отмечен функциональными перемещениями. В iOS появился Центр управления, который можно в любой момент вызвать от нижнего края экрана, а также ряд других усовершенствований.

Bluetooth, AirPlay, обмен сообщениями по разным сетям — функции, которые используются во многих приложениях.



дизайн не сводится к внешнему виду

Дизайн = внешний вид + поведение

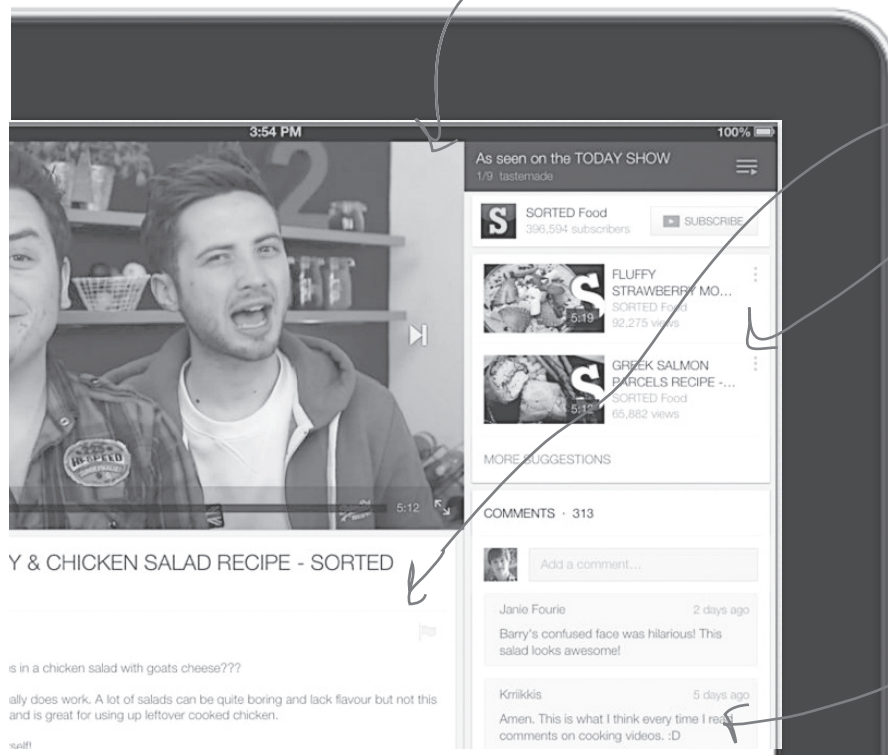
Более тонкие шрифты и повышение разрешения экрана в процессе становления экосистемы iOS означают, что новая версия iOS начинает использовать дополнительное место для вывода более подробной информации.



Обратите внимание на «плоские» значки.

Нижняя панель предназначена для размещения значков часто используемых приложений... Она была создана не по соображениям дизайна.

iPad выглядит и ведет себя иначе, потому что на планшетах используется другой интерфейс.



Для интерфейса iPad характерны сворачиваемые представления и упрощенная навигация.

Хороший дизайн не сводится к внешнему виду приложения.

Несколько представлений, интерфейсы которых управляются действиями пользователя.

Пять главных отличий iOS 7

- 1 Более плоский дизайн.**

В веб-дизайне и в iOS проявляется тенденция к переходу на плоский дизайн. Отбрасываемые тени и синие подчеркнутые ссылки старого Интернета постепенно исчезают. Теперь устройства и веб-страницы не используют тени и не пытаются имитировать визуальную глубину.
- 2 Скевоморфизм остался в прошлом. Капитально.**

В iOS 6 компания Apple рекомендовала разработчикам имитировать в интерфейсах реальные объекты. Предполагалось, что умение работать с меню или календарем поможет пользователю в работе с устройством iOS. В iOS 7 приложения стали более похожими на разных устройствах, потому что они в полной мере используют свою цифровую природу.
- 3 Унификация взаимодействий на разных устройствах.**

iPad и iPhone отличаются друг от друга, и мы еще поговорим о том, как эффективно использовать особенности каждого устройства. Но в iOS 7 компания Apple начала масштабное наступление на унификацию взаимодействий между iPhone и iPad.
- 4 Стратегическое использование цветов.**

В приложениях iOS интерактивные элементы выделяются цветом (вместо теней и кнопок), так что старомодные подчеркнутые ссылки остались в прошлом. Вместо этого текст, с которым можно взаимодействовать, отличается от статического текста цветом.
- 5 iOS создает ощущение глубины.**

Использование параллакса в iOS 7 в сочетании с датчиками устройства создает иллюзию визуальной глубины за значками приложений. Кроме того, из-за эффекта полупрозрачности в приложениях пользователь видит, как представление «скользит» под навигационными элементами приложения; так создается иллюзия визуальной многослойности.

у дальтоников могут возникнуть трудности с некоторыми цветами; заранее проанализируйте возможные проблемы.

Информация к размышлению: iPad — не iPhone

До настоящего момента мы не задумывались над тем, будет ли приложение выполняться на iPhone или iPad — код для обоих устройств был одинаковым. Однако отличия *существуют*: помимо размера экрана, iPad всегда поддерживает изменение ориентации, тогда как многие приложения для iPhone работают только в вертикальном (книжном) режиме. Другим важным фактором является продолжительность работы с устройством. Приложения iPhone проектируются для быстрого, простого доступа к информации. Пользователи iPad обычно работают с устройством в течение более продолжительных периодов времени, а их взаимодействия отличаются большим уровнем интерактивности.

В общем, вам придется по-разному относиться к этим устройствам.

Проблема не решается простым увеличением изображений и шрифтов для iPad. Во-первых, приложение будет хуже выглядеть, а во-вторых, Apple может отклонить его.



Различия между устройствами...

Под увеличительным стеклом

Посмотрите, как выглядит Календарь на каждом устройстве. Подумайте над тем, какие решения были приняты Apple... Вам придется принимать похожие решения в своих приложениях!

iPad...

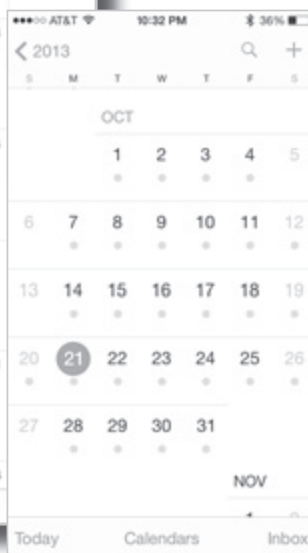


Внизу расположены важные кнопки, часто используемые в приложениях.

NIQ рекомендует эффективно использовать экранное пространство в приложениях для iPad — лишнее место заполняется дополнительной информацией.

В iPad используется полноэкранное представление. Приложение напоминает настенный календарь.

iPhone



Из-за ограниченной площади экрана выводится меньшее количество информации, но общая схема навигации остается неизменной.



Различия между устройствами... Под увеличительным стеклом

Версии Safari на iPhone и iPad очень похожи. Некоторые средства управления могут размещаться сверху или снизу, но в целом взаимодействия проходят одинаково. Впрочем, один аспект навигации отличается принципиально...



На iPad используются вкладки, как в интерфейсах портативных компьютеров.

На iPhone кнопка «>>» открывает трехмерную галерею открытых сайтов.



Очень важно своевременно понять, что ваш интерфейс просто не подходит для маленького экрана. Работать со вкладками на экране смартфона неудобно, поэтому было выбрано другое решение.

Довольно о iPad / iPhone...
Чем iOS отличается от Android?

Беседа у камина



Сегодня в студии: каково это — разрабатывать приложения для других устройств?

Разработчик Apple:

Привет. Мы вроде бы разобрались с тем, каково живется в моем мире с Apple. Интересен свежий взгляд со стороны.

Да, есть и руководство по программированию. В нем содержится много информации о новшествах в Objective-C, о стандартных взаимодействиях с пользователем и программных решениях, которые следует применять... и все такое. А на каком языке пишутся приложения Android?

Что значит — «не обязаны»? Разве ваши приложения не должны рецензироваться, чтобы попасть в магазин App Store?

Выходит... ваши приложения могут вытворять все, что захочет разработчик? Я слышал про жалобы пользователей на подозрительное поведение приложений Android...

Погоди, ты изменяешь базовое поведение устройства? А кто следит за тем, чтобы код работал на всех телефонах и планшетах на базе Android?

Разработчик Android:

Похоже, разработчикам iOS приходится выполнять множество правил. NIG, процесс рецензирования... да еще и руководство по программированию впридачу?

Все приложения Android пишутся на Java. И у нас есть много справочной информации, причем бесплатной, но мы не обязаны использовать все это, если не хотим!

Какой магазин? У нас их несколько, а не один, как у вас в iOS. Есть Google Play, есть Amazon Appstore... и наверное, будет еще больше. И никакие одобрения не нужны. Приложение просто попадает в магазин, и все. Как тебе такое?

Да, конечно, но это просто потому, что какой-то разработчик принял неудачное решение. А мои решения всегда удачны, особенно если раньше никто не делал ничего подобного. И ты не поверишь, какие трюки я порой изобретаю, чтобы добиться от устройств того, что проектировщикам оборудования *и в голову не приходило...*

Вообще-то никто... Но на моем телефоне и двух планшетах все работает!

Разработчик Apple:

Да, но ведь разных устройств Android... миллион, не меньше!

Эээ... да. Значит, ты можешь написать приложение, которое работает только на твоём телефоне и на твоём планшете? Может, процесс рецензирования Apple и выглядит странно, но зато от моих приложений устройство не впадает в кому...

Конечно. Замечательно. То есть ты можешь очень быстро выпустить своё дефектное приложение? Вот пользователи обрадуются!

Зато я уверен в том, что то, что работает у меня, будет работать и у всех остальных, фанатик Android!

Анархист!

До встречи... А я пока проверю свои продажи в App Store...

Разработчик Android:

Разве не здорово? Такой огромный выбор! А что у вас — три размера? Да у нас на порядок больше! И мы используем каждое устройство так, как хотим. Я строю своё приложение так, как считаю нужным!

Да, бывает и такое, ну и что? Я выпускаю обновление. И не надо ждать, пока какая-то дурацкая команда что-то одобрит!

Сильная сторона Android — гибкость. И ты не лезь к нам со своим разъёмом Lightning, понял?

Лемминг!

Наплевать. Ты просто... постой, мне надо бежать, снова сообщают об ошибке. Нужно выпускать версию 324...

КЛЮЧЕВЫЕ МОМЕНТЫ



- Если вы хотите создать хорошее приложение, важно соблюдать правила Human Interface Guidelines (HIG).
- Частичная потеря гибкости дизайна компенсируется тем, что даже незнакомые приложения выглядят привычно.
- Запланируйте время на проверку, рассчитывая на то, что в результате выпущенное приложение будет содержать меньше ошибок.
- Apple жестко контролирует свои приложения. Нравится вам это или нет — таков путь iOS со всеми его достоинствами и недостатками.

Часть Задаваемые Вопросы

В: Как правила HIG изменялись со временем?

О: С течением времени правила HIG становились более гибкими. В App Store появлялось все больше приложений, технические новшества открывали возможность для новых типов взаимодействий, а компания Apple пропускала эти новшества и брала их на вооружение.

Например, во многих приложениях используется элемент управления, который «выдвигается» сбоку и частично закрывает главный экран (в частности, он широко используется в Facebook). Этот элемент управления нигде не описан в HIG, но он уже может считаться вполне стандартным для приложений iOS.

В: Не приведет ли соблюдение HIG к потере индивидуальности моих приложений?

О: Да и нет. HIG существует для того, чтобы взаимодействия с вашим приложением походили на взаимодействия с другими приложениями, чтобы упростить задачу пользователя. Однако творческий подход — замечательная штука, и новые пользовательские интерфейсы помогают выделиться на фоне конкурентов. Ваш дизайн может быть оригинальным и без нарушения HIG и стандартных схем взаимодействий iOS.

В: А если я пишу игру? Какие правила действуют в этом случае?

О: Как только вы отходите от использования стандартных элементов управления, вы начинаете отходить от HIG. Когда у вас появится некоторый опыт разработки iOS, вы начнете представлять, когда следует заботиться о HIG (например, разместить настройки приложения в пакете настроек), а когда можно идти по полностью нестандартному пути.

В: Как насчет естественных пользовательских интерфейсов? Это еще актуально?

О: Когда-то было *очень* актуально. На первых порах компания Apple очень настойчиво продвигала идею имитации интерфейсов из реального мира в приложениях (особенно для iPad). Отличным примером служит приложение Календарь — неровные страницы и знакомый дизайн настенного календаря выглядят «как настоящие». Однако по мере становления платформы iOS приложения начали постепенно отходить от этой идеи.

В: Так разработчикам Android не приходится получать «добро» на распространение своих приложений?

О: Нет. Android — платформа с открытым кодом, и это сообщество не подразумевает «сторожа», добавляющего лишний уровень утверждения приложений. Впрочем, это не значит, что никаких правил нет вообще. Большинство успешных приложений Android добровольно соблюдает рекомендации. Для Android существуют такие же руководства по программированию и организации взаимодействия с пользователем, как на стороне Apple... Просто они не являются обязательными.

В: Разнообразие устройств является проблемой и для iOS?

О: Этот фактор безусловно должен учитываться при проектировании пользовательского интерфейса. Сейчас приходится рассматривать три размера: исходный экран iPhone/iPod, новый экран iPhone 5 и экран iPad. Также у каждого из этих устройств существуют разные разрешения. Вам придется все чаще применять относительное позиционирование в своих представлениях, что в конечном итоге упростит проектирование для разных устройств.

Ваш дизайн может быть оригинальным и без нарушения HIG и стандартных схем взаимодействий iOS.

У бассейна



Выловите из бассейна надписи и расставьте их по колонкам, к которым они относятся (iPhone, iPad или iPod Touch). Одна надпись может использоваться в нескольких колонках. Ваша **задача** — составить полный список функциональных возможностей iPhone, iPad и iPod Touch.

iPod Touch

iPhone

iPad

Внимание: каждый предмет из бассейна может использоваться многократно!



У бассейна. Решение



Выловите из бассейна надписи и расставьте их по колонкам, к которым они относятся — iPhone, iPad или iPod Touch. Одна надпись может использоваться в нескольких колонках. Ваша **задача** — составить полный список функциональных возможностей iPhone, iPad и iPod Touch.



Будьте
осторожны!

Этот список может
изменяться.

Apple постоянно
предлагает новые
устройства и новые возможности.

Проверьте сами!

iPod Touch

iPod

Запускается большинство приложений

Просмотр видео

Ограниченное определение местонахождения

Акселерометр

Wi-Fi

Может оснащаться камерой

Некоторую информацию о местонахождении можно узнать по данным сетей Wi-Fi.

iPhone

iPod

Запускается большинство приложений

Просмотр видео

GPS

Акселерометр

Wi-Fi

Сотовая связь

Камера

Внешний динамик

Запись видео

Магнитометр

Возможно, какие-то пункты списка оказались неожиданными, например динамик?

Спорный вопрос.

Только на 3GS и более новых моделях

iPad

iPod

Запуск приложений iPhone и iPad

Просмотр видео

Ограниченное определение местонахождения

Акселерометр

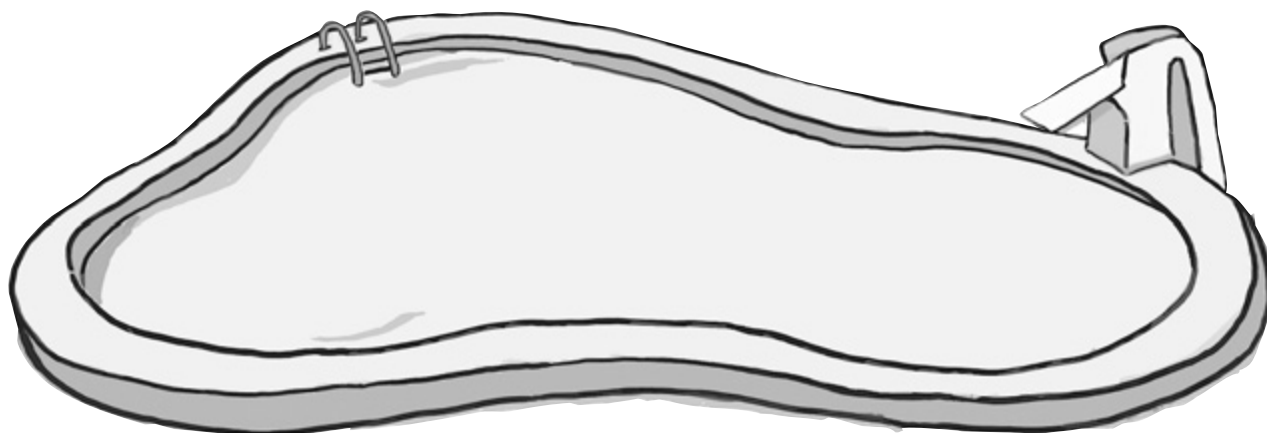
Wi-Fi

Может поддерживать GPS

Может оснащаться камерой

Внешний динамик

Запись видео





Ваш инструментарий Apple

Ваш инструментарий пополнился новыми знаниями об экосистеме Apple.

Процесс рецензирования

- Если вы хотите, чтобы ваше приложение продавалось в App Store, вам придется передать его на рецензирование.
- Планируя процесс рецензирования, выделите время на возможную повторную отправку.
- Вам придется проверять и поддерживать разные устройства и разные возможности для каждого устройства.

Не пытайтесь...

- ...обойти ограничение на покупку контента в приложениях.
- ...неправильно записывать названия любых продуктов Apple или использовать эти продукты в рекламе.
- ...занимать слишком много памяти — особенно для контента, не создаваемого пользователем.

КЛЮЧЕВЫЕ МОМЕНТЫ

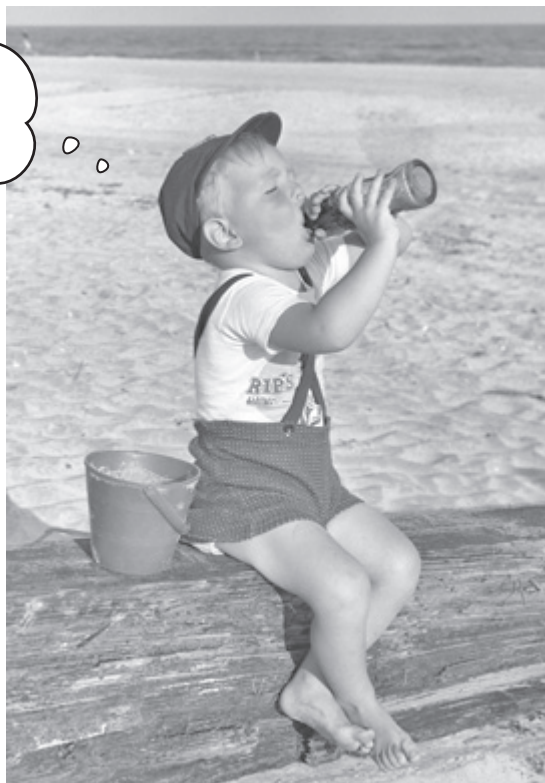


- Если вы хотите создать хорошее приложение, важно соблюдать правила Human Interface Guidelines (HIG).
- Частичная потеря гибкости дизайна компенсируется тем, что даже незнакомые приложения выглядят привычно для пользователя.
- Запланируйте время на рецензирование, но также рассчитывайте на то, что в результате выпущенное приложение будет содержать меньше ошибок.
- Apple жестко контролирует свои приложения. Нравится вам это или нет — таков путь iOS со всеми его достоинствами и недостатками.

6 core data и ячейки табличного представления

Как поймать любимую передачу

Я играю в «Остров Гиллигана». А хорошо бы снова посмотреть его по телевизору!



Устраивайтесь поудобнее и послушайте историю — историю о судьбоносном путешествии. Сегодня часто возникает одна проблема: как работать с большими объемами данных и организовать их представление в формате, подходящем для мобильных устройств? Это можно делать многими способами, включая обработку данных и представление в формате, удобном для навигации и интерпретации. Для примера возьмем программу телепередач: в эфире идет великое множество программ. Что делать поклоннику «Острова Гиллигана»?

Это Ваше приложение

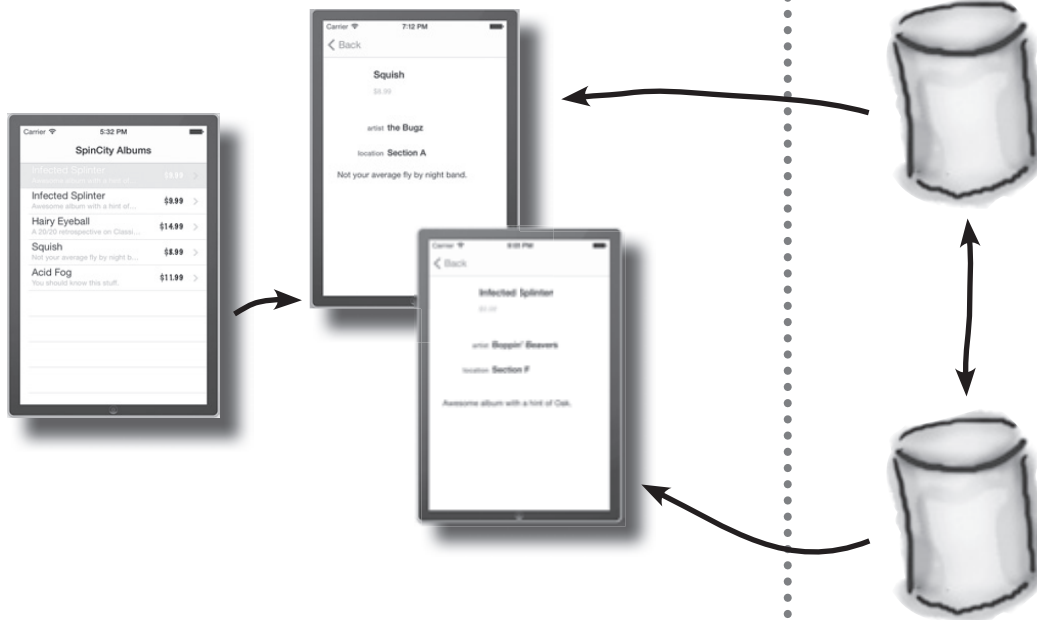
Если не считать игр и приложений для чтения, большинство приложений на мобильных устройствах предназначено для эффективного выбора и получения данных. Приложения — счетчики калорий, календари, почтовые программы, программы чтения новостей, погодные приложения — все это вариации на одну тему. До настоящего момента мы работали с простейшими способами хранения, массивами и списками `plist`.

Списки `plist` отлично подходят для хранения локальных данных, состоящих в основном из чисел и строк. Хотите сохранить фотографии или видеоролики? Изображения? Списки `plist` выглядят уже не так привлекательно. Производительность ухудшается, а передача данных с устройства происходит неэффективно. Списки `plist` представляют собой файлы XML, поэтому операции загрузки и сохранения должны выполняться со всеми данными. Списки `plist`, в отличие от баз данных, не поддерживают частичный произвольный доступ.



Это ваше приложение с данными

Многие приложения работают с большими объемами данных. Списки plist устанавливают ограничения на размеры и отношения между данными, которые могут существовать в формате plist. Чтобы расширить возможности приложения по работе с данными, придется перейти на более сложную структуру управления данными.



Нужен дополнительный уровень, с которым все это заработает.

Часто задаваемые вопросы

В: Чем база данных отличается от списка plist?

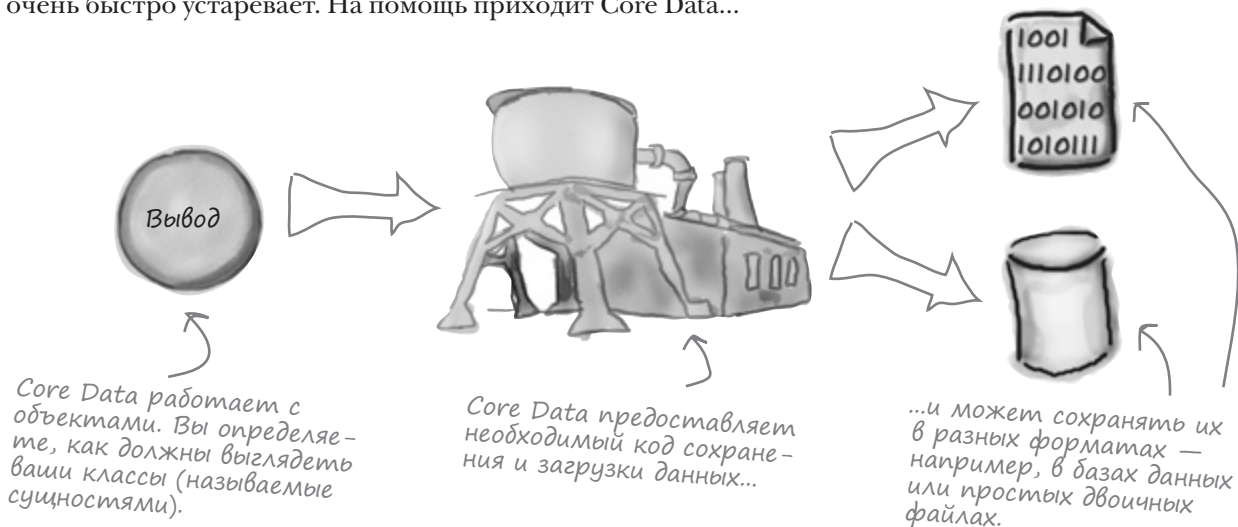
О: И базы данных, и списки plist могут использоваться для хранения данных. Список plist в действительности представляет собой файл XML, который используется для хранения простых и часто используемых типов данных. Нужно сохранить несколько чисел или простейших строк? Легко. Но когда требуется сохранить данные с возможностью произвольного доступа или поиска или если ваши данные слишком велики и не помещаются в памяти одновременно, приходится искать другое решение.

В: Как определить, какой способ следует использовать?

О: Списки plist прекрасно подходят для прототипов. Когда вы сталкиваетесь с проблемами производительности или хотите использовать возможности, которые попросту не поддерживаются, переходите на базу данных. Для работы с базой данных потребуется управляющая логика.

Знакомство с Core Data

Загрузка и сохранение данных — особенно больших объемов — играют важную роль во многих приложениях. А если вам потребуется отсортировать эти данные несколькими разными способами? Код, написанный для таких операций, очень быстро устаревает. На помощь приходит Core Data...



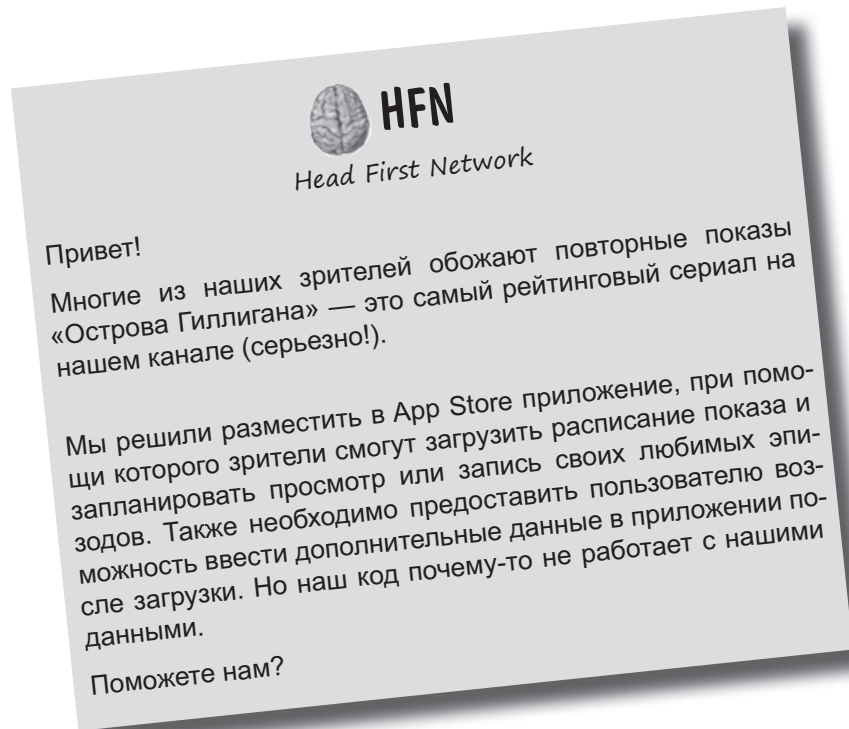
Погодите, это еще не все!

Core Data упрощает загрузку и сохранение данных, но этим дело не ограничивается. Эта проверенная временем инфраструктура, которую компания Apple перенесла из Mac OS X в iOS в версии 3.0, предоставляет следующие возможности:

- 1 Загрузка и сохранение объектов.**
 Core Data автоматически загружает и сохраняет объекты на основании описаний сущностей. Core Data даже позволяет управлять отношениями между объектами, выполнять миграцию данных при изменении версии, определять обязательные и необязательные поля и организовать проверку полей.
- 2 Различные способы хранения данных.**
 Core Data скрывает механизм фактического хранения данных от приложения. Операции чтения и записи могут выполняться как с базой данных SQLite, так и с двоичным файлом — вы лишь должны сообщить Core Data, в какой форме должны храниться ваши данные.
- 3 Управление памятью с отменой и возвратом.**
 Core Data исключительно эффективно реализует управление объектами в памяти и отслеживание изменений в объектах. Инфраструктура может использоваться для отмены и возврата, постраничного просмотра больших баз данных и т. д.

...Кстати, о данных

У канала Head First Network появилась отличная идея для нового приложения. Но похоже, у них возникли какие-то сложности.



КЛЮЧЕВЫЕ МОМЕНТЫ



Как, по вашему мнению, действовали другие разработчики? Может, вы готовы предложить какие-то изменения?

Приложение Gilligizer

Другие разработчики уже подготовили и создали представления, просто им нужно немного помочь с управлением данными. В только что созданных представлениях не выводится никакая информация. Происходит то же самое, что и в приложении SpinCity, когда в нем выводились пустые представления.





Упражнение

Постройте приложение Gilligizer. Как и в предыдущих случаях, приложение находится в GitHub, готовое к загрузке и запуску.

1

Загрузите готовый код из GitHub.

Вместо того чтобы создавать новый проект, мы скопируем его из репозитория (как это было сделано в главе 1). На экране «Welcome to Xcode» выберите вариант Checkout an Existing Project, затем вариант «Repository». В поле должен содержаться адрес <https://github.com/dpilone/Head-First-iPhone-iPad-Development-3rd-Edition.git>, который мы использовали ранее в книге.

2

Выберите нужный раздел.

Исходная версия кода каждой главы размещается в соответствующем разделе. Выберите раздел главы 6 и нажмите кнопку Next. Остается выбрать место сохранения и имя (например, chapter 6 или Gilligizer) и нажать кнопку «Checkout».

3

Протестируйте приложение.

Проект находится на вашем компьютере; постройте и запустите его. Приложение должно работать с минимумом функций!



Теперь, когда вся подготовительная работа проделана, можно переходить к данным!

Core Data начинается с данных

Как и в большинстве задач управления данными, сначала необходимо построить модель данных и интегрировать ее в приложение. Так как макет детализированного представления уже создан, мы можем воспользоваться им для определения того, какие данные необходимо включить в приложение. Пока что самым простым способом будет ручной ввод данных пользователем. Не беспокойтесь, позднее в приложение будет добавлена большая база данных с расписанием передач.





Развлечения с магнитами

Используйте то, что вы узнали из приложения SpinCity, для заполнения пропусков в данных передачи. Пока нам нужно хотя бы примерно представить, какие поля нам понадобятся.

Класс Show

```
#import <UIKit/UIKit.h>
```

```
NSNumber * episodeID;
```

```
NSNumber * desc;
```

```
NSDate * showTime;
```

```
#import <CoreData/CoreData.h>
```

```
NSNumber * showTime
```

```
NSString * title;
```

```
NSNumber * firstRun;
```

```
NSString * desc;
```



Развлечения с магнитами. Решение

Ниже показаны данные класса, которые мы посчитали нужным включить в это приложение.

Класс Show

```
NSString * title;
NSString * desc;
NSDate * showTime;
NSNumber * episodeID;
NSNumber * firstRun;
```

В iOS это называется «сущностью» (entity).

```
NSNumber * showTime
```

```
#import <UIKit/UIKit.h>
```

```
NSNumber *desc;
```

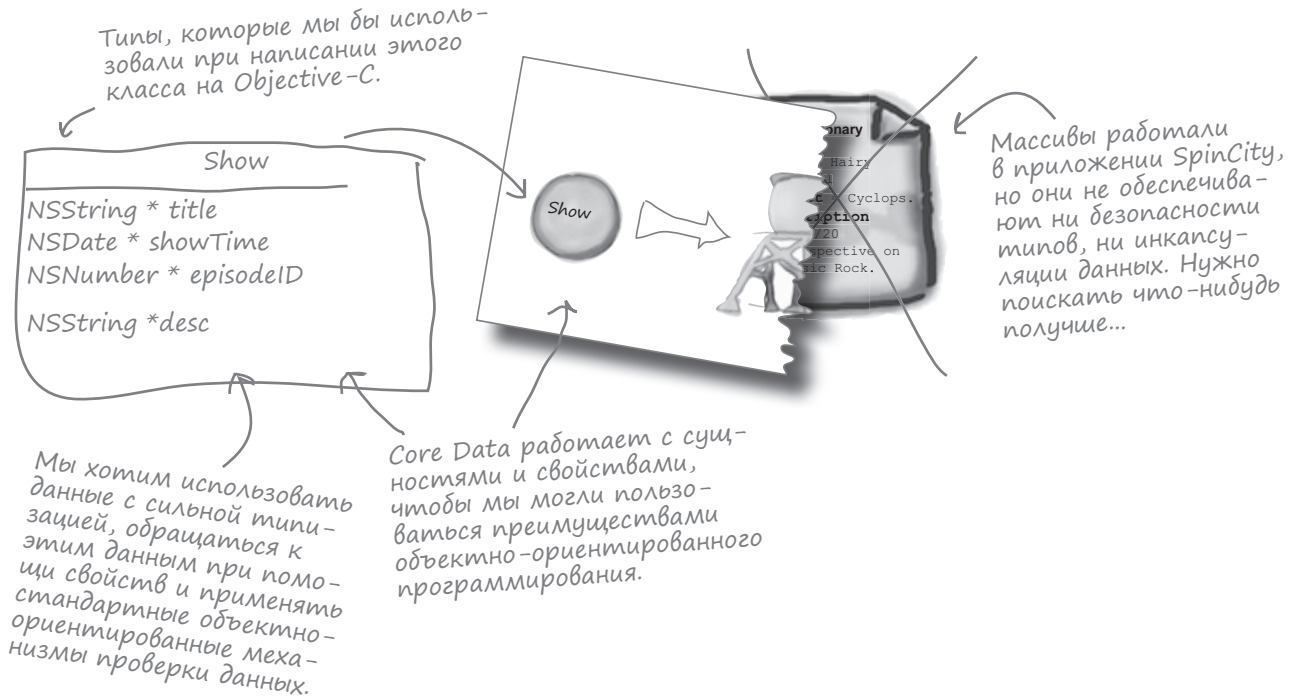
Часто задаваемые вопросы

В: Почему для идентификатора серии используется тип `NSNumber`, а не целочисленный тип?

О: Если вы заглянете в Википедию (как это сделали мы!), то увидите, что номера серий записываются в формате «сезон.серия». Такой формат весьма типичен, так что целочисленный тип здесь не подходит.

Core Data работает с сущностями

Итак, мы знаем, как должны выглядеть данные; теперь нужно разобраться, как отобразить эту структуру данных на концепции, с которыми оперирует Core Data. В Core Data для описания различных частей обработки данных используется особая терминология. **Сущность** (entity) представляет некую информацию, которую вы хотите сохранить в базе данных. В нашем случае сущности соответствуют показам «Острова Гиллигана»; они состоят из атрибутов (название серии, описание и т. д.), а для более сложных данных — отношений между сущностями.



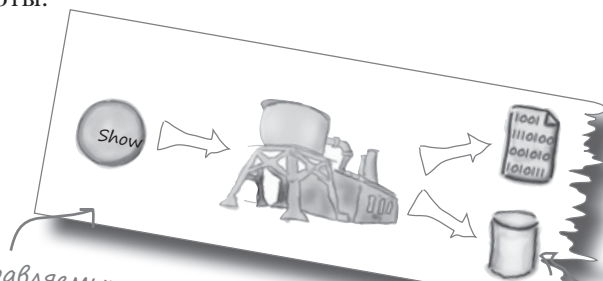
Необходимо определить сущность для Core Data

Сущности — абстрактные понятия, поэтому для них необходимо создать программное представление, которое могло бы использоваться приложением (вскоре мы поговорим об этом подробнее); но сначала их необходимо представить так, чтобы с ними можно было работать средствами Core Data.

Core Data не только предоставляет объектно-ориентированное представление данных, но и позволяет определять данные в графической форме. Впрочем, есть одна загвоздка — встроенные средства Core Data поддерживают конкретный набор типов данных, поэтому в определении сущности необходимо ограничиться типами, которые предоставляет Core Data...

Для описания сущностей Core Data использует модель управляемых объектов

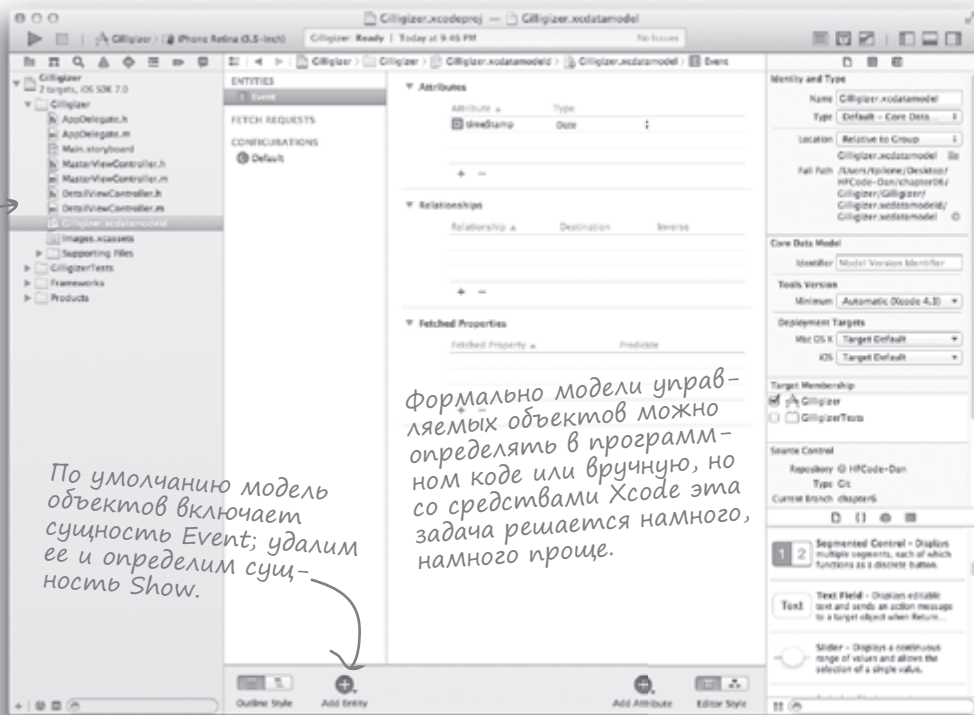
Сущности, которыми управляет Core Data, называются «управляемыми объектами» (**Managed Objects**). Описания сущностей (свойства, отношения, информация типа и т. д.) реализуются через модель управляемых объектов. Core Data обращается к модели управляемых объектов во время выполнения, чтобы определить, как выполнять загрузку и сохранение данных из хранилища (например, базы данных). Шаблон разделенного представления Xcode на базе Core Data содержит пустую модель управляемых объектов, которая станет отправной точкой для дальнейшей работы.



Наш шаблон включает пустую модель управляемых объектов с именем `Gilligizer.xcdatamodeld`. Щелкните на ней, чтобы открыть это представление.

Модель управляемых объектов описывает объекты, которые мы будем запрашивать или пытаться сохранить.

Она содержит всю информацию, необходимую Core Data для чтения и записи этих данных в хранилище.



Шаблон настроен так, что Core Data при запуске пытается загрузить все модели управляемых объектов, определенные в приложении. Нам понадобится только эта модель.

По умолчанию модель объектов включает сущность `Event`; удалим ее и определим сущность `Show`.

Формально модели управляемых объектов можно определять в программном коде или вручную, но со средствами Xcode эта задача решается намного проще.

Давайте создадим сущность `Show`...

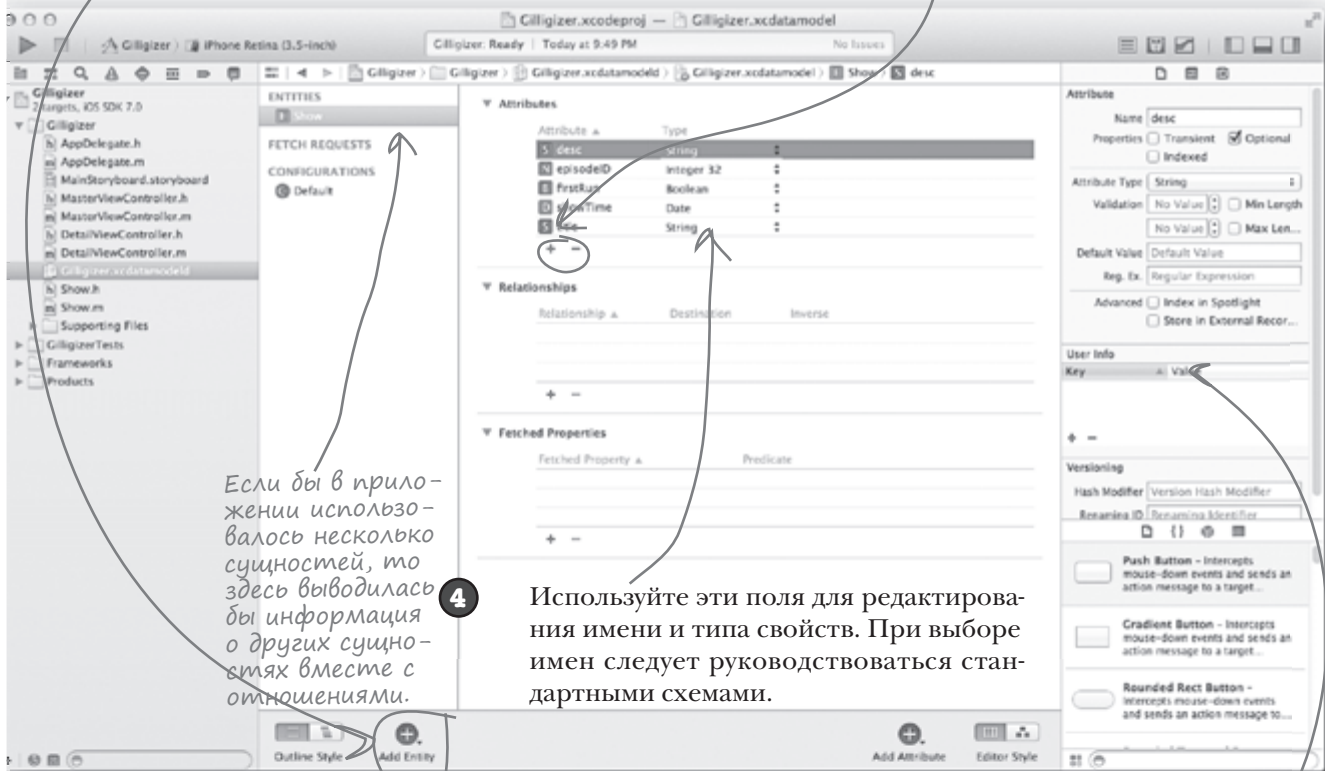
Построение сущности Show

Мы должны построить сущность Show для модели управляемых объектов. Так как Show не связывается отношениями с другими классами, остается лишь добавить свойства. Откройте файл *Gilligizer.xcdatamodeld* для создания типа данных Show.

1 Удалите автоматически сгенерированную сущность («Event»); для этого щелкните на ней и нажмите клавишу Delete.

2 Чтобы добавить сущность Show, щелкните на кнопке «+» и замените имя на «Show».

3 Когда сущность появится в списке, вы сможете добавить атрибуты в модель данных (снова при помощи кнопки «+»).



В редакторе свойств также можно устанавливать ограничения для свойств (минимум, максимум, обязательное значение и т. д.). Пока мы эти возможности не используем...

Часто Задаваемые Вопросы

В: Для чего нужны флажки `transient` и `indexed checkboxes` в Xcode при создании свойств?

О: Флажок `transient` означает, что Core Data не будет загружать или сохранять данное свойство. Временные свойства обычно используются для хранения значений, которые вычисляются только один раз (по сообщениям производительности или удобства) по данным, хранимым в сущности. При использовании временных свойств обычно реализуется метод `awakeFromFetch`, который вызывается сразу же после того, как Core Data загрузит вашу сущность. В этом методе можно вычислить значения временных свойств и задать их.

Флажок `indexed` сообщает Core Data, что инфраструктура должна попытаться создать индекс для этого свойства. Индексы используются для ускорения поиска, так что если свойство должно использоваться для поиска сущностей (идентификаторы клиентов, номера счетов и т.д.), прикажите Core Data проиндексировать его для повышения скорости поиска. Индексы занимают место и могут замедлять вставку новых данных в хранилище, поэтому использовать их следует только в том случае, если они действительно улучшают эффективность поиска.

В: Я видел константы, объявляемые с префиксом `k`. Они чем-то отличаются?

О: Нет — это стандартная схема назначения имен. Программисты C и C++ обычно записывают имена констант в верхнем регистре, а Apple обычно использует символ «`k`» нижнего регистра.

В: А если мне потребуется использовать тип, который Core Data не поддерживает?

О: Разумеется, проще всего попытаться заставить ваши данные работать с одним из встроенных типов. Если это невозможно, создайте пользовательский тип и реализуйте методы, которые будут использоваться Core Data для загрузки и сохранения значений. Наконец, вы можете сохранить данные в двоичной форме и написать код кодирования/декодирования данных во время выполнения.

В: Какие еще виды хранения данных поддерживает Core Data?

В: Core Data поддерживает три типа хранения данных в iOS: двоичные файлы, базы данных SQLite и хранение в памяти. Вариант SQLite наиболее удобен, и мы будем использовать его в приложении Gilligizer. Кстати, он используется по умолчанию. Двоичные файлы удобны своей атомарностью; в них либо успешно сохраняется все, либо не сохраняется ничего. Проблема в том, что для обеспечения атомарности iPhone при-

ходится читать и записывать весь файл при любых изменениях. В iOS двоичные файлы используются относительно редко. Данные, хранимые в памяти, на диске вообще не сохраняются, но к ним можно применять все возможности Core Data по поиску, сортировке и отмене/возврату операций.

В: Какие типы данных/табличные структуры использует Core Data при записи в базу данных SQLite?

О: Вам эта информация просто не нужна. Хотя данные записываются в базу данных SQLite, формат, типы и структуры не являются частью открытого API, и теоретически Apple может их изменить. Предполагается, что вы работаете с базой данных SQLite как с «черным ящиком» и обращаетесь к ней только через Core Data.

В: Мы получаем большой объем готового кода, но я не вижу, чтобы мы хоть на шаг приблизились к работе с массивом.

О: Мы должны были сообщить Core Data, с какой информацией работает приложение. После того как подготовительная работа завершена, можно применить ее на практике.



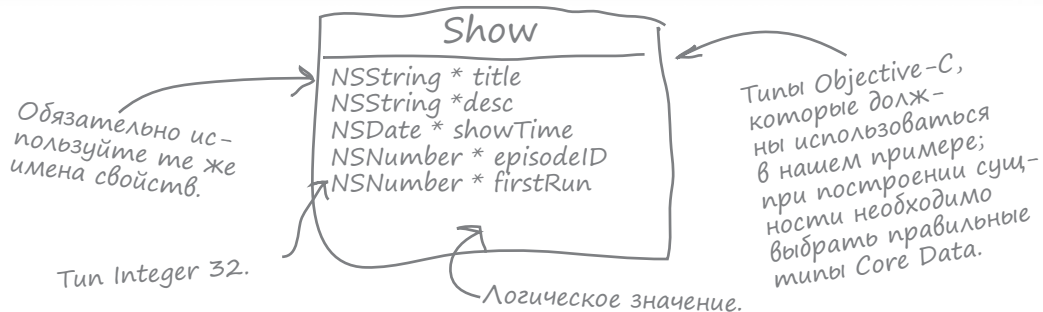
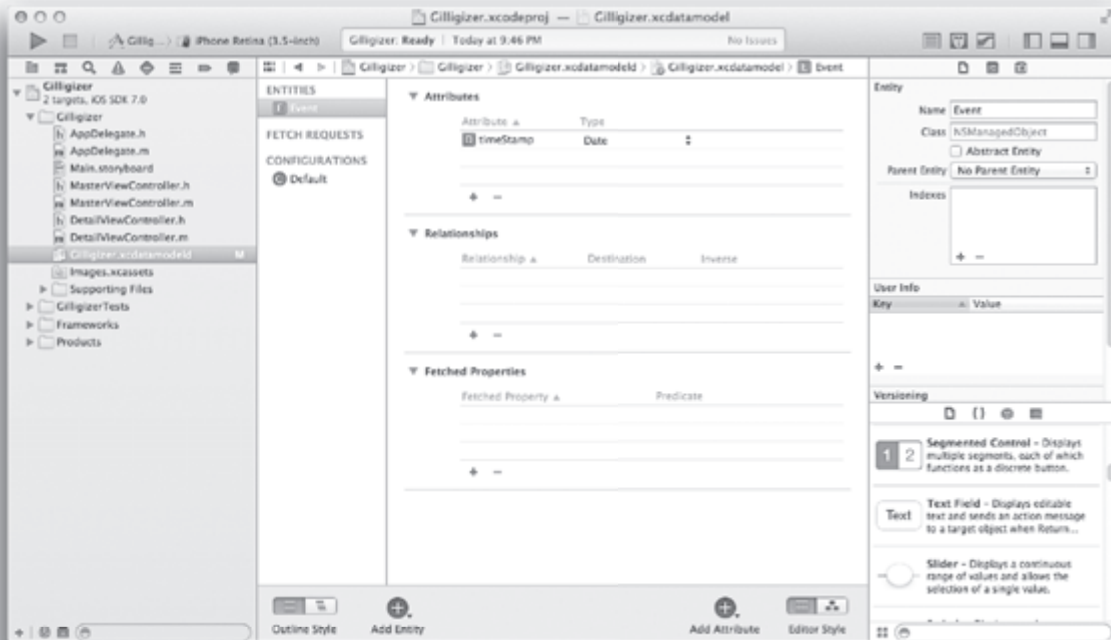
Будьте
осторожны!

Проследите за тем, чтобы ваша модель объектов ничем не отличалась от нашей!

При написании приложения существуют многочисленные возможности для настройки модели данных. Так как мы предоставим вам готовую базу данных для Gilligizer, ваша модель должна полностью совпадать с нашей!

ПОСТРОЕНИЕ МОДЕЛИ УПРАВЛЯЕМЫХ ОБЪЕКТОВ

Завершите построение сущности Show в модели управляемых объектов, руководствуясь информацией о сериях, которую мы хотим хранить. Помните: типы Core Data не будут полностью совпадать с типами Objective-C. Проследите за тем, чтобы свойствам Show были заданы имена, изображенные ниже на диаграмме.



ПОСТРОЕНИЕ МОДЕЛИ УПРАВЛЯЕМЫХ ОБЪЕКТОВ

Мы завершили построение сущности Show в модели управляемых объектов на основании информации Show, которую требуется сохранить. Не забудьте: между типами Core Data и типами Objective-C нет точного совпадения.

Проследите за тем, чтобы флажок «optional» был снят для всех свойств.



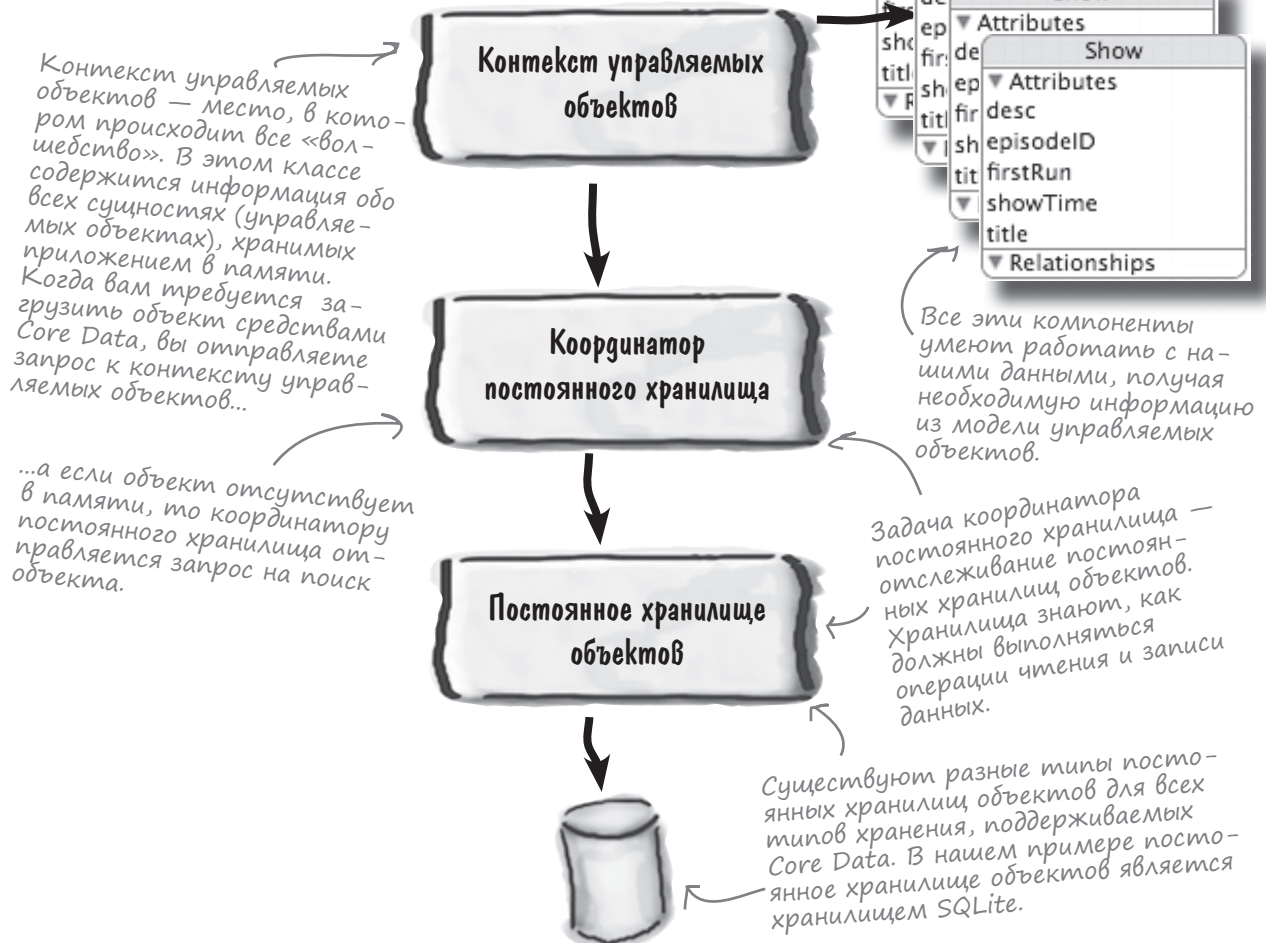


Core Data под увеличительным стеклом

Суть Core Data заключается в управлении объектами

До настоящего момента мы говорили об описании объектов для Core Data, но при этом ни слова не сказали о том, как выполнить с ними нужные операции. Для этого необходимо поглубже заглянуть во внутреннюю структуру Core Data.

Технологический стек Core Data состоит из трех важнейших блоков: **контекста управляемых объектов**, **координатора постоянного хранилища** и **постоянного хранилища объектов**.





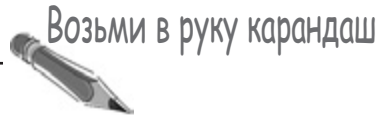
Замечательно, и как же выполнять операции загрузки и сохранения с использованием контекста управляемых объектов?

Также часто встречается сокращение МОО (Managed Object Context).

Точно!

Контекст управляемых объектов взаимодействует с базой данных для получения информации. С этой частью мы разобрались. Теперь нам нужен класс, способный взаимодействовать с МОО и преобразовать эту информацию к виду, который может использоваться приложением. Нам нужен класс Show....

**Но ведь вам нужен код?
Core Data может сделать
это за вас.**



Xcode может создать на основании модели управляемых объектов класс Show, который будет использоваться в приложении, как и любой другой класс.

- 1 Выберите файл *Gilligizer.xcdatamodel* и щелкните на сущности *Show*.**
Прежде чем приказывать Xcode сгенерировать для вас класс, необходимо выбрать сущность Core Data.
- 2 Создайте новый класс управляемого объекта...**
Выберите команду Editor→Create NSObject Subclass... Вам будет предложено выбрать место для сохранения файла. Так как файл содержит только одну сущность, Xcode не предлагает выбрать тип для сохранения.
- 3 Сгенерируйте файлы *.h* и *.m*.**
Щелкните на кнопке Create; в проект включаются файлы *Show.h* и *Show.m*. Перетащите их в группу */Gilligizer*, если они еще не находятся в ней.

Возьми в руку карандаш

Решение

Ниже показано, как в Xcode на базе модели управляемых объектов создается класс Show, который используется в приложении как любой другой класс.

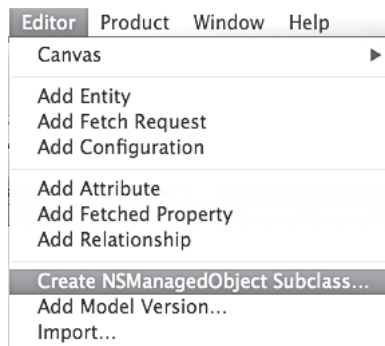
- 1 Выберите файл **Gilligizer.xcdatamodel** и щелкните на сущности **Show**.

Прежде чем приказывать Xcode сгенерировать для вас класс, необходимо выбрать сущность Core Data.



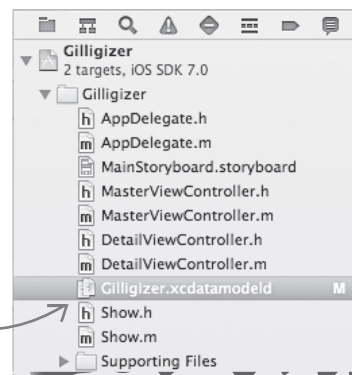
- 2 Создайте новый класс управляемого объекта...

Выберите команду **Editor→Create NSManagedObject Subclass...** Вам будет предложено выбрать место для сохранения файла. Так как файл содержит только одну сущность, Xcode не предлагает выбрать тип для сохранения.



- 3 Сгенерируйте файлы **.h** и **.m**.

Щелкните на кнопке **Create**; в проект включаются файлы **Show.h** и **Show.m**. Перетащите их в группу **/Gilligizer**, если они еще не находятся в ней.



Сгенерированный класс Show соответствует модели управляемых объектов

Xcode создает для сущности Show два новых файла: заголовочный файл *Show.h* и файл реализации *Show.m*. Откройте оба файла и посмотрите, какая информация была в них включена.

Новый класс Show является производным от *NSObject*, так как он представляет управляемый объект.

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface Show : NSObject

@property (nonatomic, retain) NSString * title;
@property (nonatomic, retain) NSString * desc;
@property (nonatomic, retain) NSDate * showTime;
@property (nonatomic, retain) NSNumber * episodeID;
@property (nonatomic, retain) NSNumber * firstRun;

@end
```

Класс содержит свойства, которые мы ожидали в нем увидеть... Но не содержит полей?!?

Для типов Core Data, выбранных в модели управляемых объектов, были подобраны соответствующие типы Objective-C.

Show.h

Часть Задаваемые Вопросы

В: Я не понимаю, откуда взялись эти типы. Тип *NSNumber* используется и для *episodeID*, и для *firstRun*?

О: При определении атрибутов модели управляемых объектов мы использовали базовые типы (*Integer32*, *Boolean*). Этот вариант хорошо подходит для модели данных, но когда Core Data преобразует сущность в управляемый объект, мы должны использовать объекты для определения типов данных. Так как Objective-C поддерживает как базовые типы, так и объекты, Core Data упаковывает некоторые базовые типы в классе *NSNumber*.



И кода здесь тоже нет... Но вероятно, вы скажете, что и об этом мне не нужно беспокоиться?

Точно! Инфраструктура Core Data позаботится об этом.

Класс `Show.m` практически пуст, и вместо синтеза свойств они объявляются с новой директивой `@dynamic`.

Реализация класса `Show` практически пуста!

```
#import "Show.h"
@implementation Show

@dynamic title;
@dynamic desc;
@dynamic showTime;
@dynamic episodeID;
@dynamic firstRun;

@end
```



Show.m

Часто Задаваемые Вопросы

В: Постойте: если класс придется генерировать повторно, все мои изменения пропадут. Что с этим делать?

О: Это верно. Если у вас имеется собственный код, который для вас важен, есть два варианта: создайте категорию для управляемого объекта или subclassируйте его.

NSManagedObject также реализует свойства

Новая директива `@dynamic` сообщает компилятору, что он не должен беспокоиться о методах чтения и записи, необходимых для свойств. Однако эти методы должны откуда-то появиться, иначе во время выполнения при попытке обратиться к этим свойствам произойдет ошибка. Здесь на помощь снова приходит `NSManagedObject`. Так как класс `NSManagedObject` управляет памятью полей, на базе которых работают свойства, он также предоставляет реализации методов чтения и записи на стадии выполнения. Реализация этих методов классом `NSManagedObject` предоставляет ряд других преимуществ:

- 1 Класс `NSManagedObject` знает, когда изменяются свойства, может проверить новые данные, а также оповестить об изменениях другие классы.
- 2 `NSManagedObject` может отложить получение информации о свойстве до того момента, когда его кто-то запросит. Например, эта возможность используется в отношениях с другими объектами.
- 3 `NSManagedObject` отслеживает изменения свойств и предоставляет поддержку отмены/возврата.

Вы получаете все это, не написав ни одной строки кода!



А теперь остается лишь приказать
Core Data загрузить Show...



Готово к употреблению

Мы включили в проект GitHub дополнительный код, который упростит работу над этой частью. Если вы загрузили проект из репозитория, то увидите, что код закомментирован, чтобы вам было проще следить за изложением. Так как вы уже один раз занимались заполнением детализированного представления, мы решили предоставить вам этот код в готовом виде.

```
(void)configureView
{
    // Обновление пользовательского интерфейса
    // для элемента детализированного представления.
    if (self.detailItem) {
        self.titleField.text = [self.detailItem valueForKey:@"title"];
        self.episodeIDField.text = [NSString stringWithFormat:@"%d",
        [[self.detailItem valueForKey:@"episodeID"] integerValue]];
        self.descriptionView.text = [self.detailItem
        valueForKey:@"desc"];
        self.firstRunSegmentedControl.selectedSegmentIndex = [[self.
        detailItem valueForKey:@"firstRun"] boolValue];
        self.showTimeLabel.text = [[self.detailItem
        valueForKey:@"showTime"] description];
    }
}
```

Мы просто заполня-
ем пользовательский
интерфейс данными
из объекта Show.

Мы используем 'boolValue' или
'integerValue', когда свойство является
экземпляром NSNumber, но мы знаем
(из описания сущности), что в дейст-
вительности должен использоваться
базовый тип.



DetailViewController.m



Готово к употреблению

```
#pragma mark - Editing

- (IBAction)textFieldEditingChanged: (id) sender
{
    if ([sender isEqual:self.titleField]) {
        [self.detailItem setValue:self.titleField.text forKey:@"title"];
    }
    else {
        [self.detailItem setValue:[NSNumber numberWithInt:[self.
episodeIDField.text integerValue]] forKey:@"episodeID"];
    }
}
```

Если пользователь вводит данные в текстовом поле названия, необходимо обновить поле title объекта Show.

Здесь пользователь обновляет идентификатор серии. Как и в предыдущем случае, перед присваиванием в объекте Show данные необходимо упаковать в NSNumber.



DetailViewController.m

```
- (IBAction)newEpisodeValueChanged: (id) sender
{
    [self.detailItem setValue:[NSNumber numberWithBool:self.
firstRunSegmentedControl.selectedSegmentIndex] forKey:@"firstRun"];
}

#pragma mark - UITextFieldDelegate

- (void)textViewDidChange:(UITextView *)textView
{
    [self.detailItem setValue:textView.text forKey:@"desc"];
}

@end
```

Когда пользователь изменяет состояние переключателя новой серии, следует обновить логическое поле firstRun в объекте Show.

Пользовательский интерфейс использует класс UITextView для вывода описания серии. Мы реализуем метод UITextViewDelegate, чтобы получать оповещения каждый раз, когда пользователь обновит описание.



DetailViewController.m



Джим: Наши заказчики нервничают...

Фрэнк: Ну, у нас есть модель...

Джо: ...и представление из GitHub.

Джим: И еще есть кнопка добавления, верно? Они хотят, чтобы пользователь мог добавить информацию о показе, а потом просмотреть ее.



Фрэнк: И как работает эта кнопка?

Джим: Просто загляни в код!

Джо: Как насчет метода вставки нового объекта? Я видел его в файле *MasterViewController.m*, когда мы добавляли остальной код.

```

- (void)insertNewObject:(id)sender
{
    NSManagedObjectContext *context = [self.fetchedResultsController
managedObjectContext];

    // Вставка новой сущности Show.
    Show *show = [NSEntityDescription insertNewObjectForEntityForName:@"Show"
inManagedObjectContext:context];
    show.desc = @"On this episode...";
    show.title = @"New Episode";
    show.firstRun = [NSNumber numberWithBool:FALSE];

    NSInteger count = [[self.fetchedResultsController sections][0]
numberOfObjects];
    show.episodeID = [NSNumber numberWithInt:count];
    show.showTime = [NSDate dateWithTimeIntervalSinceNow:(86400 * count)];

    // Сохранение контекста.
    NSError *error = nil;
    if (![context save:&error]) {
        // Замените эту реализацию кодом обработки ошибки.
        // Вызов abort() заставляет приложение выдать сообщение о сбое
        // и завершиться. Не используйте эту функцию в окончательной
        // версии, но во время разработки она может пригодиться.
        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }
}

```

Класс `NSEntityDescription` используется инфраструктурой Core Data для поиска сущности в модели управляемых объектов и вставки нового управляемого объекта в МОС.

Так как все атрибуты были объявлены обязательными, необходимо задать значения по умолчанию.

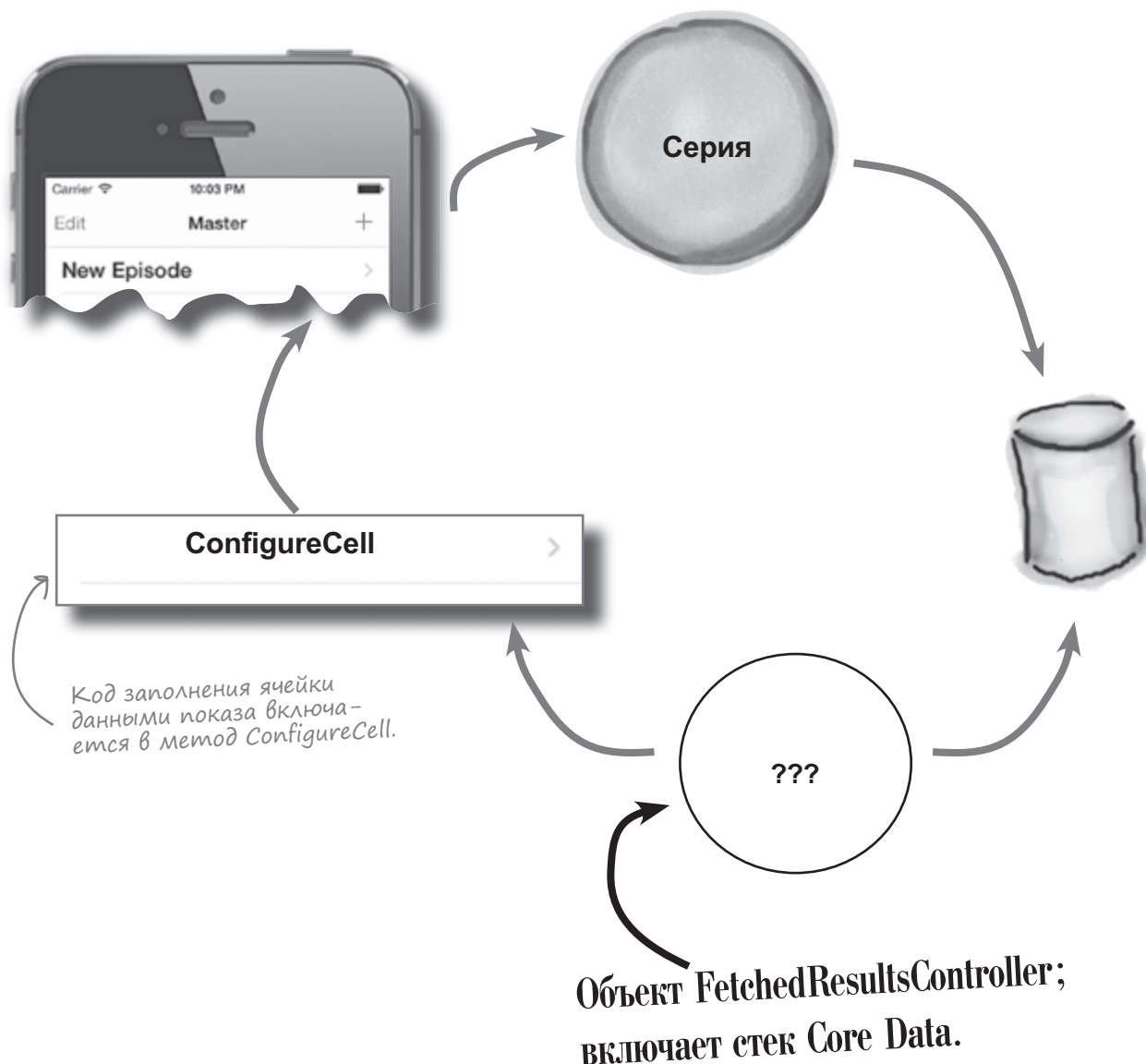
Здесь мы приказываем МОС сохранить информацию нового объекта Show в базе данных. Если объект Show недействителен, попытка сохранения завершится неудачей.



MasterViewController.m

У нас есть объект... Теперь его необходимо вывести

У нас имеется все необходимое для создания нового объекта Show и его добавления в базу данных. Но это не значит, что он автоматически появится на экране! Теперь нам понадобится код выборки чтения объекта и его вывода в табличном представлении.



Вывод сущности в приложении Gilligizer

Чтобы вывести объект Show на экран, необходимо получить нужный объект и извлечь из него отображаемую информацию. Наше табличное представление сообщит, какой объект Show нам нужен, по значению indexPath соответствующей строки. Остается получить данные показа из FetchedResultsController и настроить ячейку по значениям свойств Show.

Мы получаем объект Show для этой строки таблицы из FetchedResultsController. После получения объекта остается приказать ячейке вывести название серии.

```
- (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath *)indexPath
{
    NSManagedObject *object = [self.fetchedResultsController
    objectAtIndex:indexPath];
    cell.textLabel.text = [[object valueForKey:@"title"] description];
}
```



MasterViewController.m

Часто Задаваемые Вопросы

В: Как эти данные заносятся в базу?

О: Кнопкой +, которую мы создали. При нажатии этой кнопки мы создаем новый экземпляр сущности Show, после чего сохраняем обновленный объект ManagedObjectContext. А пока пользователю (или вам во время тестирования) придется добавлять информацию вручную. Вскоре вы узнаете, как добавить в проект базу данных с тестовой информацией.

В: Откуда Core Data знает, какой объект создать в методе 'insertNewObject:'?

О: Помните, что говорилось ранее о различиях между Entity и NSManagedObject? Когда мы хотим создать новый объект Show, мы приказываем классу NSEntityDescription вставить новый объект с именем «Show». Класс ищет в модели управляемых объектов сущность с именем «Show». Обнаружив искомую сущность, он инициализирует экземпляр subclasses NSManagedObject, соответствующий сущности Show.



Мы создаем сущности в базе данных... и выводим объекты из базы данных. Но как получить эти объекты?

Легко! Приложение должно выполнить выборку результатов из базы данных.

Стек Core Data содержит большой объем кода, который понадобится нам для выборки данных из базы. Немного покопавшись в коде, вы найдете нужный метод...



Готово
к употреблению

Мы собираемся реализовать метод чтения для экземпляра `fetchResultsController`. Если экземпляра еще не создан, в этом методе мы инициализируем его.

```
#pragma mark - Fetched results controller

- (NSFetchResultsController *)fetchResultsController
{
    if (_fetchResultsController != nil) {
        return _fetchResultsController;
    }

    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
    // Изменить имя сущности по обстоятельствам.
    NSEntityDescription *entity = [NSEntityDescription entityForName:@"Show"
        inManagedObjectContext:self.managedObjectContext];
    [fetchRequest setEntity:entity];

    // Присвоить размеру пакета подходящее число.
    [fetchRequest setFetchBatchSize:20];
```

Если экземпляр FRC уже существует, мы просто возвращаем его.

Здесь FRC передается имя сущности, которую нужно получить из базы данных. Стоит обратить внимание на использование `NSEntityDescription`.





Готово к употреблению

```
// Изменить ключ сортировки по обстоятельствам.
NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"showTime" ascending:YES];
NSArray *sortDescriptors = @[sortDescriptor];

[fetchRequest setSortDescriptors:sortDescriptors];

// Изменить путь ключа имени секции и имя кэша по обстоятельствам.
// nil вместо пути ключа имени секции означает "без секций".
NSFetchedResultsController *aFetchedResultsController =
[[NSFetchedResultsController alloc] initWithFetchRequest: fetchRequest
managedObjectContext:self.managedObjectContext sectionNameKeyPath:nil
cacheName:@"Master"];
aFetchedResultsController.delegate = self;
self.fetchedResultsController = aFetchedResultsController;

NSError *error = nil;
if (![self.fetchedResultsController performFetch:&error]) {
    NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
    abort();
}

return _fetchedResultsController;
}
```

Дескриптор сортировки обеспечивает получение информации из базы данных в заранее определенном порядке. Полезно для вывода данных в таблице!



MasterViewController.m

Мы инициализируем экземпляр `NSFetchedResultsController` и приказываем ему выполнить выборку (`performFetch`). Этот вызов использует запрос выборки, созданный несколькими строками выше, для выборки и сортировки информации из базы данных.



Готово к употреблению

Так как выводимыми в таблице данными управляет `NSFetchedResultsController`, необходимо предоставить механизм, при помощи которого FRC сможет оповещать нас об изменении данных.

```
(void) controllerWillChangeContent: (NSFetchedResultsController *) controller
{
    [self.tableView beginUpdates];
}

(void) controller: (NSFetchedResultsController *) controller didChangeSection: (id
<NSFetchedResultsSectionInfo>) sectionInfo
        atIndex: (NSUInteger) sectionIndex forChangeType: (NSFetchedResultsChangeType) type
{
    switch(type) {
        case NSFetchedResultsChangeInsert:
            [self.tableView insertSections:[NSIndexSet indexSetWithIndex:sectionIndex] withRowAnimation:UITableViewRowAnimationFade];
            break;

        case NSFetchedResultsChangeDelete:
            [self.tableView deleteSections:[NSIndexSet indexSetWithIndex:sectionIndex] withRowAnimation:UITableViewRowAnimationFade];
            break;
    }
}
```

Наш класс реализует протокол `NSFetchedResultsControllerDelegate` и использует его методы для обновления табличного представления.

Наш объект FRC вызывает этот метод каждый раз, когда он обнаруживает в данных изменения, отражающиеся на секции таблицы.

Объекту FRC даже хватает способностей для передачи информации о типе внесенных изменений (вставка или удаление). Остается лишь приказать нашему табличному представлению вставить или удалить секцию.



MasterViewController.m

```

- (void)controller:(NSFetchedResultsController *)controller didChangeObject:(id)anObject
  atIndexPath:(NSIndexPath *)indexPath forChangeType:(NSFetchedResultsControllerChangeType)type
  newIndexPath:(NSIndexPath *)newIndexPath
{
    UITableView *tableView = self.tableView;

    switch(type) {
        case NSFetchedResultsControllerChangeInsert:
            [tableView insertRowsAtIndexPaths:@[newIndexPath] withRowAnimation:UITableViewRowAnimationFade];
            break;

        case NSFetchedResultsControllerChangeDelete:
            [tableView deleteRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationFade];
            break;

        case NSFetchedResultsControllerChangeUpdate:
            [self configureCell:[tableView cellForRowAtIndexPath:indexPath]
              atIndexPath:indexPath];
            break;

        case NSFetchedResultsControllerChangeMove:
            [tableView deleteRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationFade];
            [tableView insertRowsAtIndexPaths:@[newIndexPath] withRowAnimation:UITableViewRowAnimationFade];
            break;
    }
}

- (void)controllerDidChangeContent:(NSFetchedResultsController *)controller
{
    [self.tableView endUpdates];
}

```

В этом методе содержится большая часть «волшебства». Он вызывается при изменении одного из объектов, находящихся под управлением FRC.

Как и в случае с секциями, FRC сообщает нам о том, какого рода изменения были внесены в объект. И по аналогии с секциями нам остается лишь обновить табличное представление, чтобы оно соответствовало представляемым данным.

Вызывается, когда контроллер завершает оповещение о внесенных изменениях. Здесь мы сообщаем табличному представлению о том, что процесс обновления закончен.

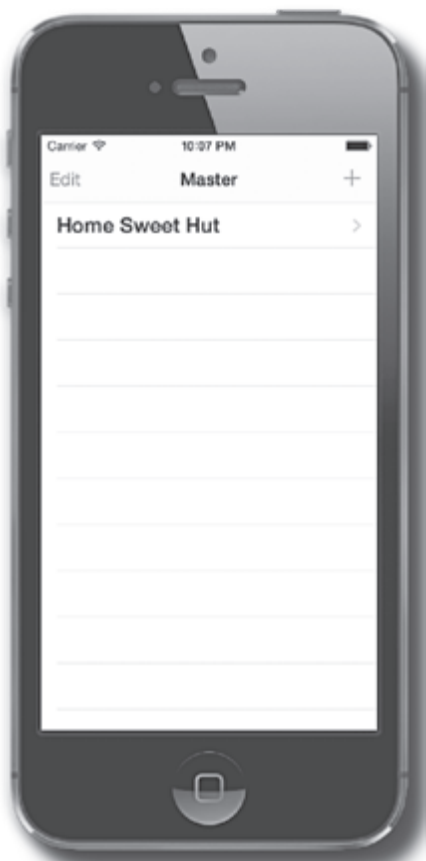




ТЕСТ-ДРАЙВ

Проверьте приложение в деле. Для надежности обязательно удалите приложение из эмулятора и проведите очистку в Xcode (Shift+Command+K).

Нажав кнопку +, вы переходите к детализированному представлению и сможете добавить текст.



Введите текст и выйдите из детализированного представления; введенный текст появляется в главном представлении.



Смех, да и только! Сначала мы генерируем модель. Теперь вся эта возня с контроллером. Что вообще происходит?

Работа с результатом выборки — непростое дело. Но вскоре дойдет и до этого... Ждите более подробных объяснений!





Ваш инструментарий Core Data

Глава 6 осталась позади, а ваш инструментарий пополнился принципами использования Core Data и ячеек табличных представлений.

Core Data

- ☐ Загружает и сохраняет объекты.
- ☐ Может использовать для хранения данных SQLite и двоичные файлы.
- ☐ Упрощает управление памятью, операции отмены и возврата.
- ☐ Использует концепцию сущностей.
- ☐ Сущностью называется абстрактное представление данных.

Технологический стек Core Data

- ☐ Модель управляемых объектов используется для описания сущности в Core Data.
- ☐ Включает контекст управляемых объектов, координатор постоянного хранилища и постоянное хранилище объектов.

Выборка данных

- ☐ `configureCell` — метод, в котором данные загружаются в представление.
- ☐ FRC (Fetch Results Controller) — код, предоставляемый Core Data для упрощения получения результатов из базы данных.

7 реализация поиска с core data

Поиск информации

Где же передача, которую мы ищем? Должен быть более удобный способ...



Просто просматривать данные уже недостаточно. Наступила эпоха больших данных, и с обычным просмотром далеко не уйдешь. Вероятно, объем данных на вашем телефоне еще не измеряется в петабайтах, и все же он достаточно велик, чтобы функции сортировки и фильтрации упростили работу с ними. Core Data содержит встроенные средства покорения больших объемов данных, и в этой главе вы научитесь пользоваться ими!

Приложение работает, но его возможности ограничены...

Первая бета-версия отправилась заказчику, но в ней не реализована вся запланированная функциональность. Приложение способно хранить и выводить информацию о сериях, а пользователи могут вводить собственные описания. Но когда в приложении накапливается большой объем данных, начинаются проблемы.

Представьте: перед вами гигантский список телепередач, и вы пытаетесь найти в нем одну, нужную вам....



РАССЛАБЬТЕСЬ

Сейчас мы займемся изучением контроллера результатов выборки.

В конце предыдущей главы был приведен значительный объем заранее подготовленного кода, использующего FRC. Сейчас мы все объясним, честно!



Джим: Мы должны спроектировать какой-то интерфейс, чтобы пользователь мог получить доступ к этим функциям.

Фрэнк: Дельная мысль. По каким полям будет производиться сортировка?

Джо: Время и дата.

Фрэнк: Что, всего два варианта? Как насчет сегментного элемента управления, это будет разумно?

Джим: Хорошо. Тогда одна часть сегмента будет выполнять сортировку по времени, а другая по дате?

Фрэнк: И для каждого сегмента надо будет создать для `FetchResultsController` новый дескриптор сортировки. Один для даты, другой для времени.

Джо: А потом мы изменяем запрос на выборку для `FetchResultsController`, приказываем выполнить выборку заново и перезагрузить таблицу.

Фрэнк: По-моему, нормально...



Упражнение

Перейдите в Xcode, разместите сегментный элемент управления и организуйте сортировку. На самом деле это несложно...

Здесь будет находиться сегментный элемент управления.

- 1 **Создайте сегментный элемент управления, о котором говорят Джим, Фрэнк и Джо.**

В главном файле раскадровки добавьте сегментный элемент управления и обновите текст сегментов на «Show Time» и «Title». Добавьте в файл `MasterViewController.m` действие и ссылку `IBOutlet` для сегментного элемента управления.

- 2 **Измените код `FetchResultsController` и реализуйте метод `segmentControlValueChanged` в `MainViewController.m`.**

Ключ сортировки `FetchResultsController` должен обновляться в зависимости от состояния сегментного элемента управления, после чего таблица обновится.



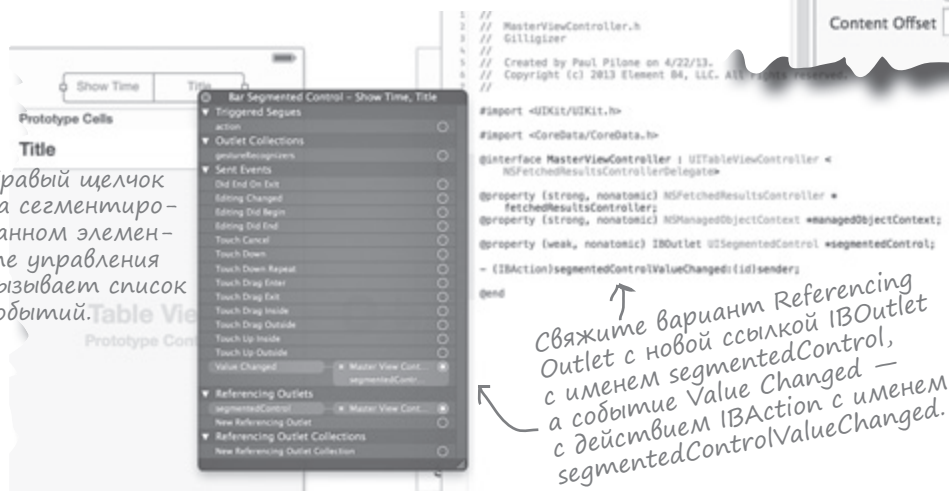
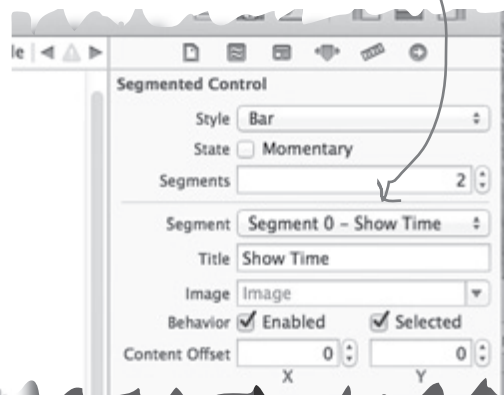


Разместите сегментный элемент управления и организуйте сортировку. Стоит вам освоить класс `NSFetchedResultsController`, и вы поймете, что отлично справляетесь с подобной работой.



Перетащите сегментный элемент управления на панель навигации контроллера главного представления.

Отредактируйте текст двух сегментов (segment 0 и segment 1), чтобы на сегментах выводились надписи Show Time и Title соответственно.



Правый щелчок на сегментированном элементе управления вызывает список событий.

Свяжите вариант Referencing Outlet с новой ссылкой IBOutlet с именем segmentedControl, а событие Value Changed — с действием IBAction с именем segmentedControlValueChanged.



Упражнение Решение

```
- (NSFetchResultsController *)fetchResultsController
{
    ...
    // Измените ключ сортировки по обстоятельствам.
    NSString *sortKey = [self.segmentedControl selectedIndex] == 0 ?
    @"showTime" : @"title";
    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
    initWithKey:sortKey ascending:YES];
    NSArray *sortDescriptors = @[sortDescriptor];

    [fetchRequest setSortDescriptors:sortDescriptors];
    ...
}
```

Измените команду создания ключа сортировки в коде `fetchResultsController`, чтобы в нем использовались атрибуты `showTime` или `title` наших сущностей Core Data (в зависимости от состояния сегментного элемента управления).



MasterViewController.m

```
- (IBAction)segmentedControlValueChanged: (id) sender
{
    self.fetchResultsController = nil;
    [self.tableView reloadData];
}
```

Реализуйте действие сегментного элемента управления с уничтожением старого объекта `FetchResultsController` и инициацией перезагрузки таблицы. При этом создается новый объект `FetchResultsController` с подходящим ключом сортировки.



MasterViewController.m



Погодите... Вы же обещали объяснить,
что там происходит с выборкой
и контроллерами...

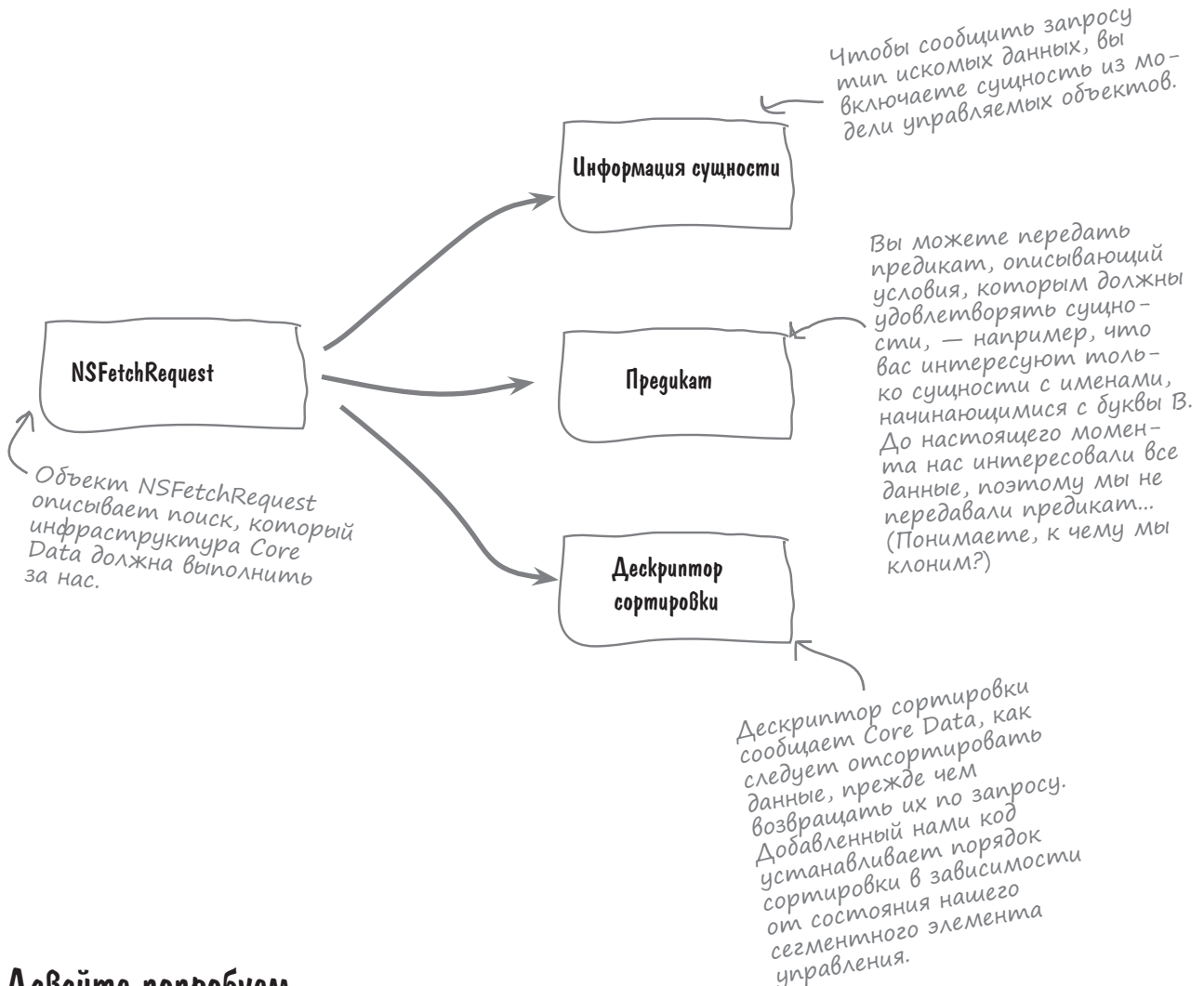
Верно, обещали.

По правде говоря, мы немного забежали вперед. Впрочем, ничего особенно сложного здесь нет.

Таблица заполняется точно так же, как и другие таблицы, которые использовались ранее. Вызывается метод `cellForRowAtIndexPath`, который должен вернуть `UITableViewCell`. Мы берем объект и начинаем настраивать его. Здесь-то и используется контроллер результатов выборки. Класс `NSFetchedResultsController` спроектирован с поддержкой `UITableView`. Остается только сообщить ему, как найти нужные данные...

Используйте `NSFetchRequest` для описания поиска

Чтобы сообщить классу `NSFetchedResultsController`, что именно мы ищем, необходимо создать объект `NSFetchRequest`. Класс `NSFetchRequest` описывает объекты, которые должны быть возвращены в результате выборки, условия отбора (например, передачи, которые начинаются до 10 часов утра), и способ сортировки возвращаемых результатов. Да, все это он делает за нас.

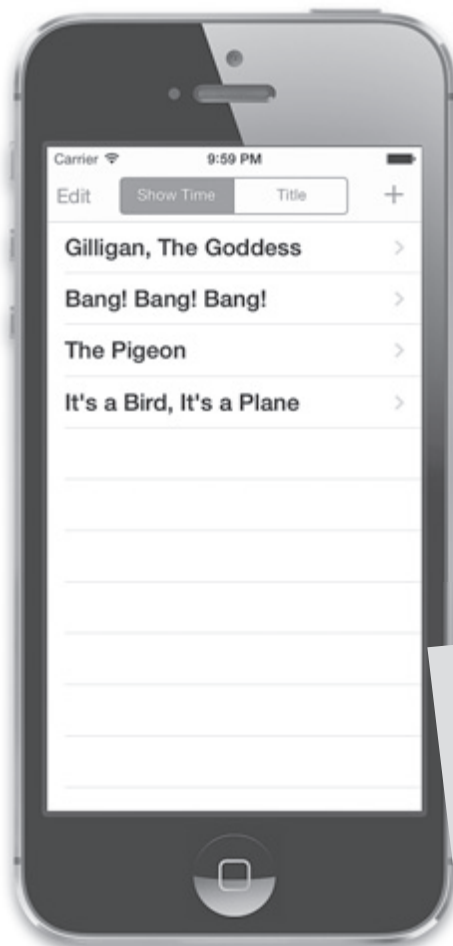


Давайте попробуем...



ТЕСТ-ДРАЙВ

Работает! Запустите приложение и введите данные нескольких эпизодов. Переключение сегментного элемента управления изменяет дескриптор сортировки, и таблица обновляется соответствующим образом. Отличная работа!



Письмо от заказчика...



HFN

Head First Network

Привет!

Спасибо за проделанную работу — выглядит здорово, но функциональность ограничена. Мы хотим, чтобы наши пользователи могли выполнять:

- Сортировку данных по времени или дате.
- Поиск в данных по ключевым словам.

В текущей версии при большом количестве передач трудно найти нужную серию или информацию о ней. Поиск и сортировка упростят жизнь пользователям.

Спасибо!



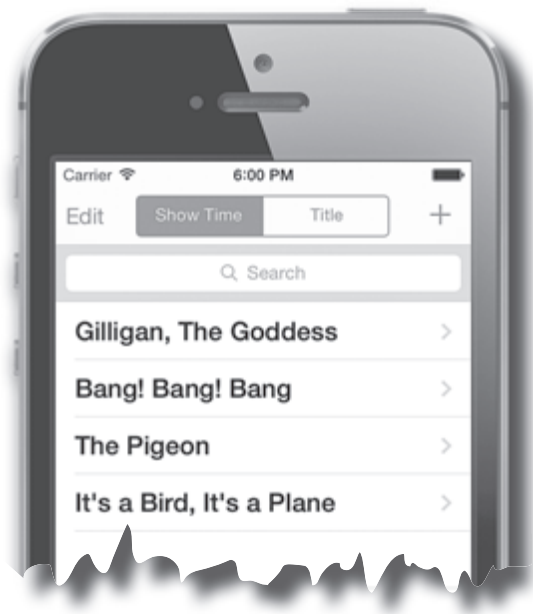
Мы будем использовать разные предикаты поиска — так же, как использовали разные дескрипторы сортировки?

Точно!

Поисковый элемент управления включает собственное табличное представление, и мы предоставим ему собственный объект `FetchResultsController`, но на этот раз мы будем изменять предикат по мере ввода данных пользователем.

В iOS 7 поддержка поиска встроена в Core Data и UIKit

Поиск в данных приложения является настолько распространенной задачей, что iOS 7 и Core Data предоставляют практически все необходимое для его организации. Как почти во всех предыдущих случаях, визуальное представление информации (UIKit) можно отделить от непосредственного поиска в данных (Core Data). Начнем с части, относящейся к UIKit.



SearchDisplayController делает почти все необходимое

iOS 7 включает компоненты `SearchBar` и `SearchDisplayController`, которые предоставляют все необходимые классы `UIComponent`. Просто включив контроллер в раскладовку, вы автоматически получаете панель поиска и контроллер, предоставляющий модальное табличное представление, заполненное результатами. От вас потребуется лишь создать необходимые связи, как и при работе с обычными табличными представлениями.



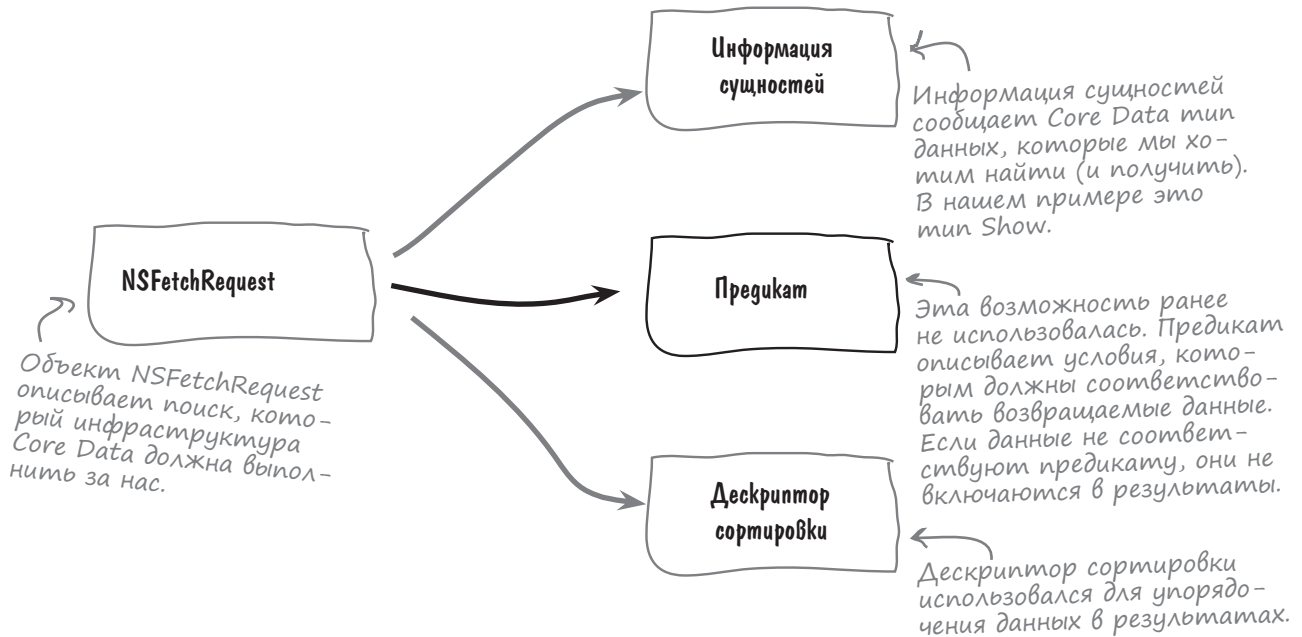
Значит, этот контроллер поиска может выполнять поиск и в наших данных?

Нет, зато контроллер результатов может!

Контроллер результатов не ограничивается одним дескриптором сортировки. Мы можем использовать расширенные средства поиска и фильтрации Core Data с классом `NSFetchedResultsController`, чтобы ограничить состав возвращаемых данных. И именно это делает элемент `UISearchDisplayController`.

Использование предикаторов для фильтрации данных

В большинстве языков баз данных **предикаты** используются для ограничения поиска и выборки только тех данных, которые соответствуют заданному критерию. Помните класс `NSFetchRequest`, о котором говорилось ранее в этой главе? Тогда мы использовали информацию сущностей и дескриптор сортировки, но предикаты были не нужны... до этого момента.



Концепции `NSFetchRequest` очень близки к традиционным концепциям SQL

Язык SQL используется для управления базами данных.

Три основные концепции `NSFetchRequest` почти идентичны компонентам выражений в стандартном синтаксисе SQL. Вот как выглядел бы наш запрос:

```
SELECT * FROM SHOWS WHERE title LIKE "Gil%" ORDER BY title ASC
```

Annotations for the SQL query:

- Under `SHOWS`: "Аналог нашей информации сущностей — может, и не полный, но близкий." (Analog of our entity information — maybe not complete, but close.)
- Under `LIKE "Gil%"`: "Предикат SQL..." (SQL predicate...)
- Under `ORDER BY title ASC`: "Условие сортировки в команде SQL." (Sorting condition in the SQL command.)

Ограничения поиска упаковываются в объект `NSPredicate` и передаются `NSFetchRequest`. Давайте немного поговорим о предикате, прежде чем связывать все воедино.

Предикат `NSFetchRequest` определяет возвращаемые данные

`NSPredicate` — обманчиво простой класс для выражения логических ограничений для `NSFetchRequest`. Ограничения выражаются с использованием имен сущностей и атрибутов, а также операторов сравнения. Простейший объект `NSPredicate` со строковым форматом условия, сходным с `NSString`, создается так:

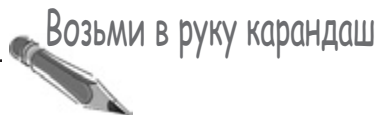
```
fetchRequest.predicate = [NSPredicate predicateWithFormat:@"%title
contains[cd] %@", searchText];
```

Но `NSPredicates` не ограничивается простыми сравнениями атрибутов. Apple предоставляет несколько субклассов (таких, как `NSComparisonPredicate`, `NSCompoundPredicate` и `NSExpression`), а также сложную грамматику для поиска по шаблону, обхода графа объектов и т. д. Даже существует возможность построения сложных запросов на выборку графическим способом в Xcode. В документации iOS имеется целое руководство по созданию предикатов для самых разнообразных задач поиска.

Панель поиска позволяет узнавать о происходящих событиях через делегата

После включения в раскладку объекта `searchDisplayController` Xcode автоматически связывает контроллер со свойством `SearchDisplayController`. Как и почти любому элементу управления iOS, панели поиска можно назначить делегата, который будет оповещаться о текущих событиях.

Необходимо сделать так, чтобы приложение реагировало на изменения панели поиска. Контроллер поиска спрашивает, что ему делать дальше, вызывая метод `shouldReloadTableForSearchString` при изменении текста поиска. Если метод возвращает `YES`, контроллер поиска обновляет свои результаты. Вопрос в том, как в действительности выполняется поиск?



Возьми в руку карандаш

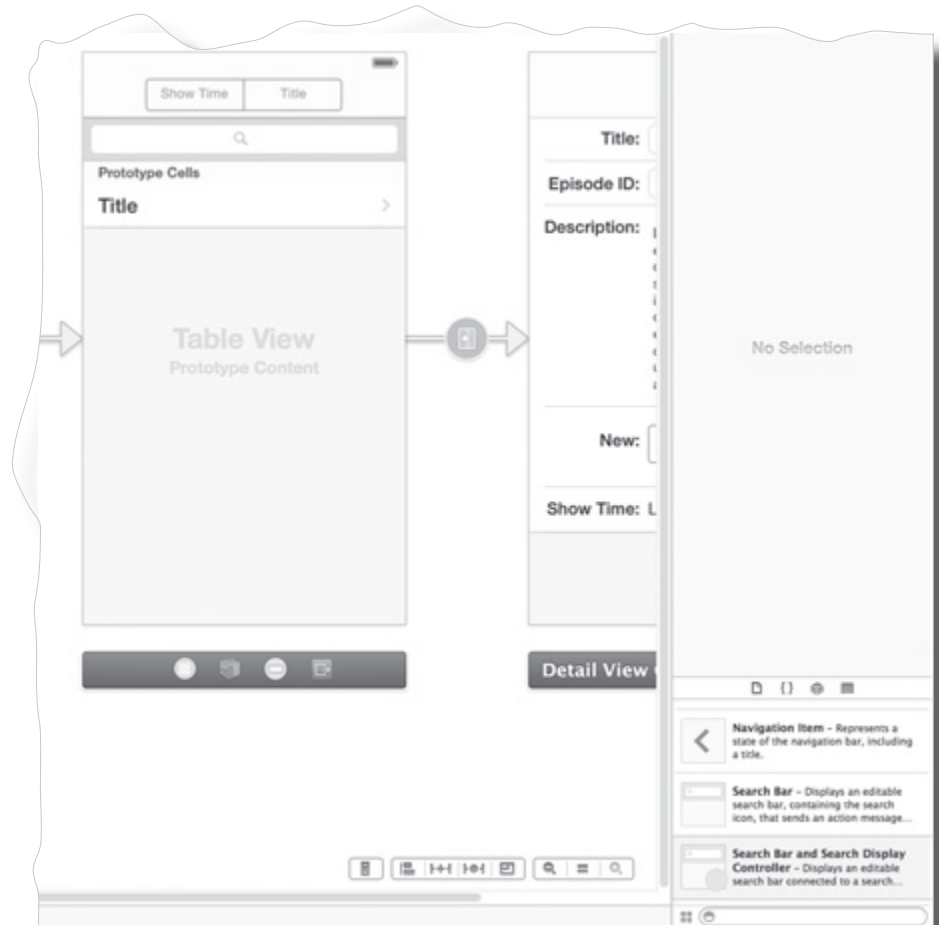
Как вы думаете, что делать дальше? Ну постарайтесь! Что нужно сделать, чтобы объект `SearchDisplayController` смог выполнять поиск в базе данных Show за нас?

→ Ответ на с. 274



Упражнение!

Перетащите в раскладовку панель поиска и SearchDisplayController. Добавьте компонент «Search Bar and Search Display Controller» в сцену главного представления в раскладовке. Компонент следует разместить непосредственно над заголовком секции Prototype Cells. Панель поиска закрепляется у верхнего края табличного представления, а контроллер вывода включается в список контроллеров у нижнего края сцены.





Возьми в руку карандаш

Решение

Вот что нужно сделать, чтобы объект `SearchDisplayController` смог выполнять поиск в базе данных `Show` за нас.

Мы можем использовать объект `NSFetchedResultsController` для выполнения поиска по тексту на панели поиска. Тот же объект `NSFetchedResultsController` для этого не подойдет!

Это был вопрос с подвохом...

Контроллеру поиска незачем выполнять поиск за нас! Компонент «Search Bar and Search Display Controller» содержит удобное текстовое поле, которое можно использовать для поиска, и модальный контроллер, который выводит табличное представление. Помните, как работает наше текущее табличное представление? Каждый раз, когда ему требуется заполнить строку данными, оно запрашивает ячейку для этой строки. Мы берем объект ячейки, заполняем его правильной информацией и возвращаем. Откуда взять информацию для строки? Из `NSFetchedResultsController`. Само табличное представление не имеет никакого отношения к способу получения данных или тому, какие данные при этом просматриваются.

Та же схема может использоваться и в этом случае:

- 1 При изменении панели поиска объект `SearchDisplayController` спрашивает нас, должен ли он обновить свое табличное представление. На этот вопрос можно вернуть `YES`, чтобы он перезагрузил данные в таблице.
- 2 При попытке обновления табличное представление начинает требовать строки. Мы используем специальный объект `NSFetchedResultsController` для получения результатов, соответствующих текущему содержимому панели поиска.



НАПРЯГИ МОЗГИ

Итак, вы в общих чертах представляете, что нужно делать; просмотрите следующий список. Выберите то, что нам понадобится для подключения `SearchDisplayController`. Учтите, что правильный ответ состоит из нескольких пунктов!

- ☐ Реализовать протокол `UISearchDisplayDelegate`.
- ☐ Реализовать метод — `(BOOL)searchDisplayController:(UISearchDisplayController *)controller shouldReloadTableForSearchString:(NSString *)searchString`, чтобы сообщить контроллеру поиска, что он всегда должен обновлять таблицу.
- ☐ Добавить новый объект `NSFetchedResultsController`, настраиваемый по результатам поиска.
- ☐ Добавить новый объект `tableView` для отображения результатов.
- ☐ Создать новую сущность для хранения результатов поиска.
- ☐ Следить за тем, какой объект `tableView` отображается на экране, чтобы мы знали, что делать в других методах.
- ☐ Создать новый контроллер детализированного представления для вывода детализации при выборе элемента результатов поиска.
- ☐ Создать свойство для `tableView` с результатами поиска.
- ☐ Создать свойство для нового объекта `searchResultsFetchedController`.



Решение НАПРЯГИ МОЗГИ

Итак, вы в общих чертах представляете, что нужно делать; просмотрите следующий список. Выберите то, что нам понадобится для подключения **SearchDisplayController**. Учтите, что правильный ответ состоит из нескольких пунктов!



Реализовать протокол `UISearchDisplayDelegate`.

Чтобы мы могли получать оповещение при изменении текста поиска.



Реализовать метод — (BOOL) `searchDisplayController:(UISearchDisplayController *)controller shouldReloadTableForSearchString:(NSString *)searchString`, чтобы сообщить контроллеру поиска, что он всегда должен обновлять таблицу.

Чтобы мы могли перенастроить `FetchedResultsController` при изменении текста поиска. Старый контроллер поиска следует уничтожить, чтобы при перезагрузке таблицы использовался новый.



Добавить новый объект `NSFetchedResultsController`, настраиваемый по результатам поиска.

Он будет храниться в свойстве, которое для этого нужно создать (см. ниже), и на основании текста поиска для него будет назначаться предикат.



Добавить новый объект `tableview` для отображения результатов.

Это берет на себя `SearchDisplayController`.



Создать новую сущность для хранения результатов поиска.

Здесь, как и в главном табличном представлении, загружаются данные `Show`.



Следить за тем, какой объект `tableview` отображается на экране, чтобы мы знали, что делать в других методах.

Мы должны знать, что отображается на экране — главное табличное представление или табличное представление результатов поиска. В частности, это необходимо для того, чтобы мы знали, какой контроллер результатов поиска должен использоваться.



Создать новый контроллер детализированного представления для вывода детализации при выборе элемента результатов поиска.

Нам придется настроить метод `prepareForSegue`, чтобы получать правильную строку из правильного табличного представления в зависимости от того, что сейчас отображается на экране. Тем не менее мы работаем только с сущностями `Show`, а наш текущий контроллер детализированного представления с ними справится.



Решение НАПРЯГИ МОЗГИ



Создать свойство для tableview с результатами поиска.

Этим занимается объект `SearchDisplayController`, у которого мы можем получить объект табличного представления.



Создать свойство для нового объекта `searchResultsFetchedController`.

Объект нужно сохранить, чтобы создавать его заново (и проводить поиск по новой) только при изменении текста, по которому проводится поиск.

Как насчет старых методов `FetchedResultsController`?
Разве их не надо обновить?



Да! Очень правильное замечание!

Нужно пройти по разным методам `MasterViewController.m`, относящимся к обновлению табличных представлений, и убедиться в том, что они обновляют правильное табличное представление, а мы используем правильный экземпляр `NSFetchedResultsController`.



А как было бы замечательно, если бы мне не пришлось вводить весь этот код в куче разных мест... Но это, конечно, только мечты.

А вот и нет!
Загрузите итоговую версию кода из репозитория Git. Вы уже знаете, что нужно делать...



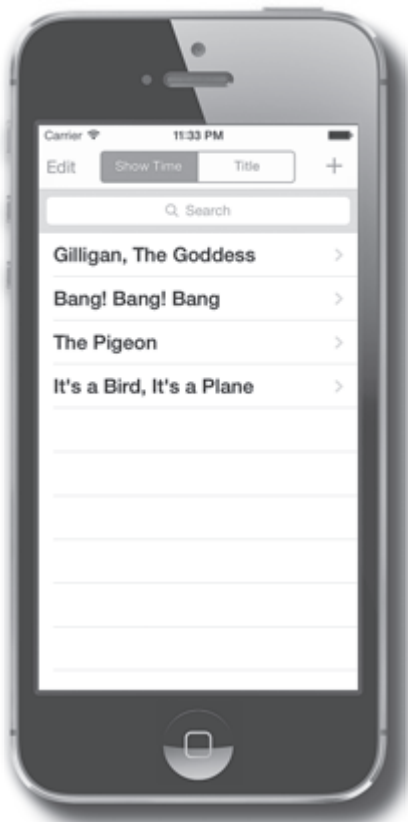
Упражнение!

Чтобы получить итоговую версию кода, вернитесь и загрузите код из основной ветви (вместо кода главы 7) — у вас окажется законченный проект.



ТЕСТ-ДРАЙВ

Загрузите итоговую версию кода этой главы. Откройте его в Xcode и проверьте, как он работает!





HFN

Head First Network

Кому: Гениальному разработчику

Потрясающе! Мы в восторге от нового приложения. Теперь каждый, кто захочет посмотреть «Остров Гиллигана», может запросто получить необходимую информацию.

Спасибо!



Ваш инструментарий поиска

Глава 7 осталась позади. В ней ваш инструментарий пополнился средствами поиска и выборки результатов.

Выборка

- ☐ Контроллер результатов выборки используется для сортировки данных.
- ☐ Чтобы выполнить выборку, необходимо выдать запрос с описанием интересующих вас объектов.
- ☐ Предикаты описывают условия, по которым осуществляется выборка.

NSFetchRequest

- ☐ Использует практически те же концепции, что и SQL.

Поиск

- ☐ Core Data и UIKit поддерживают поиск с сохранением архитектуры MVC.
- ☐ Контроллер поиска не может выполнять поиск в данных; это делает контроллер результатов.
- ☐ Компонент SearchDisplayController использует текстовое поле и модальное представление для поиска.

8 core data, map kit и core location

В поисках телефонной будки

А пока не подарили модный iPhone, приходится пользоваться этим!



Пришло время заняться расширенными возможностями.

Устройства на базе iOS содержат массу полезных встроенных возможностей. И iPhone, и iPad — это одновременно и компьютер, и библиотека, и фотоаппарат, и видеокамера, и GPS-навигатор. Область технологий геопозиционирования еще только развивается, но уже сейчас она предоставляет много полезных возможностей. К счастью, iOS позволяет относительно легко использовать эти функции оборудования.

Все старое становится ^{клевым} новым

Go Retro Unlimited

Нам нужна ваша помощь!

Компания Go Retro Unlimited обожает все старое... все клевое... и все незаслуженно забытое. Еще тогда, когда большинство людей хранили свои старые записи в коробках из-под обуви, мы превратили свои коллекции в плейлисты iTunes.

Сейчас современная культура уничтожает еще один исторический феномен: телефонные будки! Люди покупают новые крутые iPhone, а телефонные будки, которые все мы знали и любили, безжалостно сносят! Но мы можем навечно сохранить память о них!

Мы хотим хранить на своих iPad и iPhone фотографии телефонных будок... со всех концов света! Будки из Сан-Франциско, из Сиднея, из Дарфура! Мы хотим видеть фотографии и географические данные. И надеемся, что вы нам в этом поможете... Нам нужно приложение, которое может запустить любой владелец iPhone или iPad, увидевший телефонную будку. Он фотографирует объект, отправляет фотографию нам, и что самое важное — выполняет геопривязку. Только подумайте — в скором времени вы и ваши друзья сможете обмениваться фотографиями телефонных будок через iPhone пятого поколения. Это так... символично!

Работа над приложением уже началась, но нам нужна ваша помощь. Посетите GitHub и посмотрите, что было сделано. Это не так много... собственно, поэтому нам и нужна ваша помощь! Беритесь за программирование, а мы пока поищем ближайшую телефонную будку.

Монетки не найдется?

Джимми Уэйн,

директор

Да, в Дарфуре действительно есть телефонные будки. Кто устоит перед соблазном увидеть их на фото?



НАПРЯГИ МОЗГИ

Загрузите приложение из GitHub и проверьте, что уже сделано. Затем можете браться за работу!

Откройте раздел главы 8 и выберите проект RetroPhoneHunter.

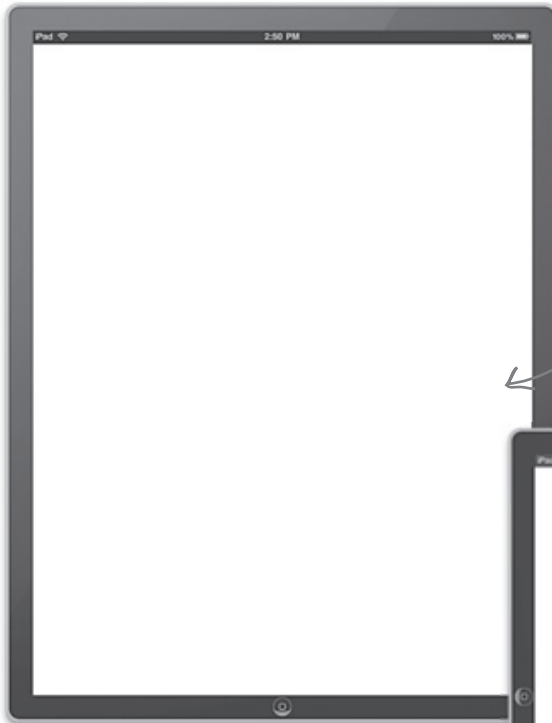
Приложение, iPad и телефонные будки

Над приложением из GitHub определенно придется потрудиться. Для каждой телефонной будки в приложении должна храниться фотография и несколько информационных полей: город, в котором находится будка, почтовый индекс, а также место, в котором фотограф может ввести заметки... Да, нам предстоит многое добавить в это приложение.



Возьми в руку карандаш

В этом виде приложение выглядит довольно уныло. Нарисуйте предполагаемый макет — где, на ваш взгляд, должны выводиться разные виды информации, сохраняемой с каждой фотографией? Скоро мы добавим и сами фотографии...



Вы можете уже сейчас планировать поддержку как вертикальной, так и горизонтальной ориентации устройства. Нарисуйте, как должно выглядеть представление в обоих вариантах.

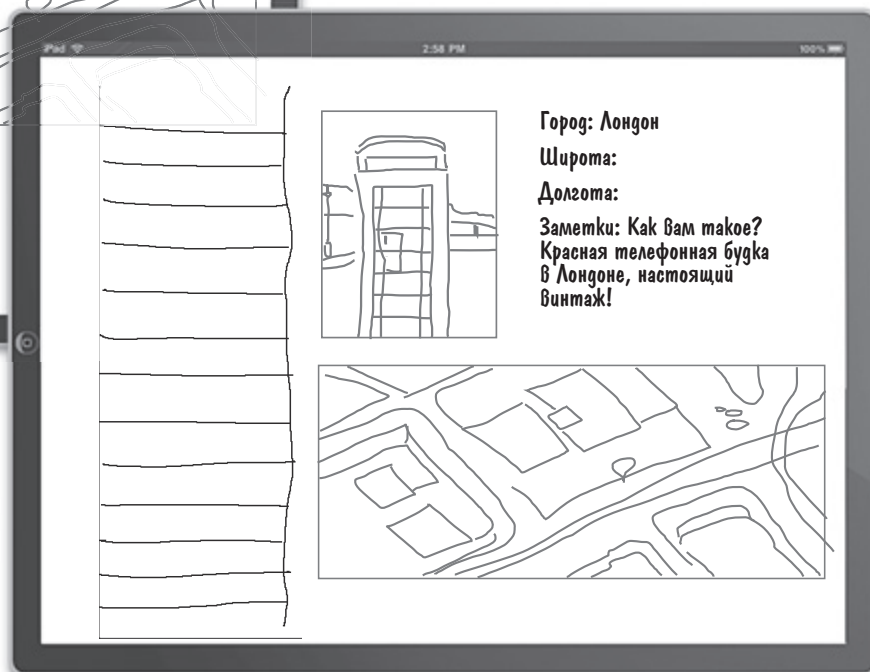
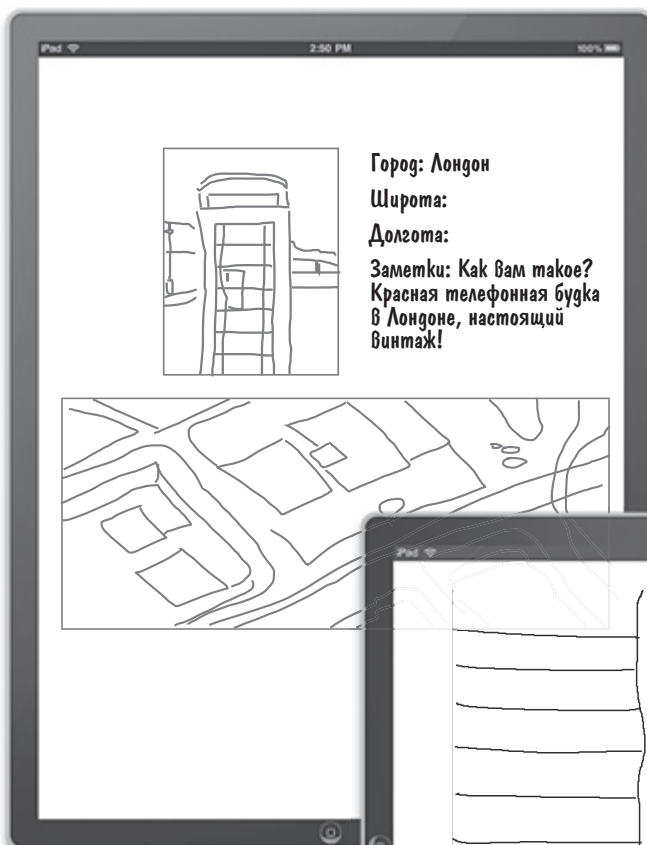


Возьми в руку карандаш



Решение

Что у вас получилось? Ниже представлены наши эскизы. Помните, что вы можете разработать для своего приложения тот дизайн, который вам нравится. Главное — чтобы вы понимали, почему было принято то или иное решение.





Длинные упражнения

Начнем с фотографий. Заказчик вряд ли будет впечатлен эскизами дизайна (и даже работоспособным приложением), если в них не будет фотографий телефонных будок. Итак, отложите карандаши и запустите редактор кода. Вот что вам предстоит сделать...

1 Свяжите приложение с базой данных, чтобы получить описания.

Это делается примерно так же, как в приложении SpinCity. Приложение, загруженное из GitHub, содержит только шаблонный код, который необходимо настроить для работы с нашим приложением.

2 Сгенерируйте класс PhoneBooth.

Создайте субкласс NSObject (так же, как это было сделано в приложении Gilligizer): выберите команду «Editor», затем «Create NSObject Subclass». Выберите сущность PhoneBooth, подтвердите сохранение сущности в проекте, затем щелкните Create.

Пока не думайте о том, как мы будем делать снимок. Эта возможность будет добавлена позднее...

3 Отредактируйте шаблонный код, чтобы вместо сущности по умолчанию Event в реализации MasterViewController.m использовалась новая сущность PhoneBooth.

Отредактируйте файл и замените ссылки «timestamp» на «name», импортируйте файл Phonebooth.h, замените в методе didSelectRowAtIndexPath класс NSObject на Phonebooth, а в вызов setValue для данных включите @"NewPhoneBooth".

4 Постройте макет детализированного представления и добавьте код, обеспечивающий его работу.

Сначала в файл раскадровки следует добавить необходимые элементы: UIImageView, надпись Name и текстовое поле, надпись City и текстовое поле, надпись Notes и текстовое поле для заметок.



Решение длинных упражнений

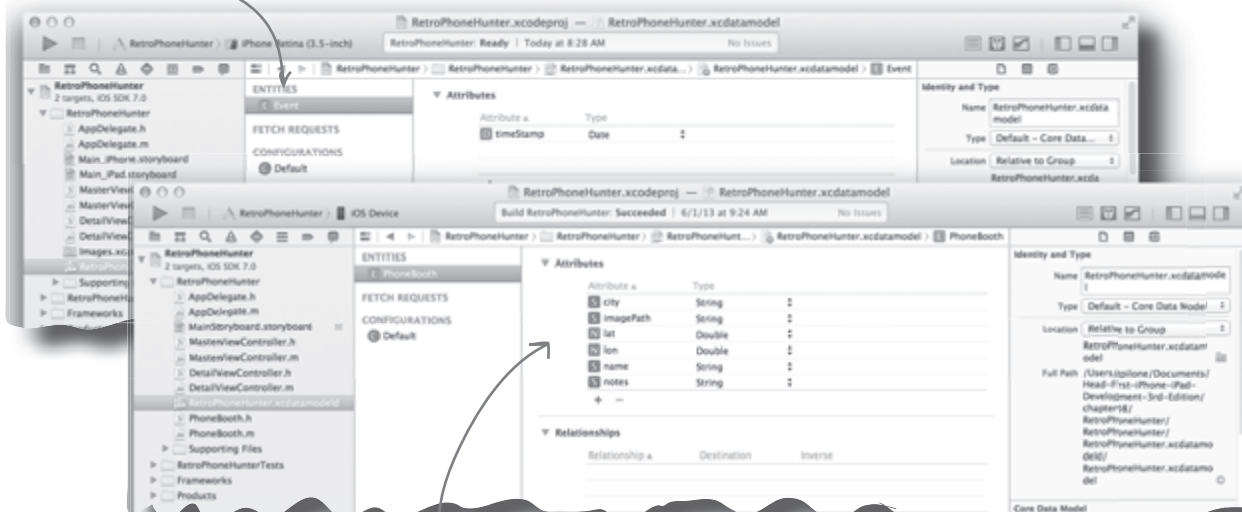
Упражнение не особенно сложное, хотя вам и пришлось написать довольно существенный объем кода. Убедитесь в том, что у вас получилось работоспособное приложение, а построение проходит без ошибок. Если у вас возникнут проблемы, обратитесь к описанию соответствующего шага и посмотрите, что получилось у нас.

1

Свяжите приложение с базой данных, чтобы получить описания.

Это делается примерно так же, как в приложении SpinCity. Приложение, загруженное из GitHub, содержит только шаблонный код, который необходимо настроить для работы с нашим приложением.

Удалите сущность по умолчанию "event". Самый простой способ — выделить ее и нажать клавишу Delete.

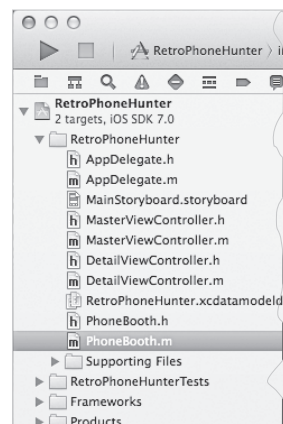


Так должна выглядеть модель, которую мы строим.

2

Сгенерируйте класс PhoneBooth.

Создайте субкласс NSObject (так же, как это было сделано в приложении Gilligizer): выберите команду «Editor», затем «Create NSObject Subclass». Выберите сущность PhoneBooth, подтвердите сохранение сущности в проекте, затем щелкните Create.



3

Отредактируйте шаблонный код, чтобы вместо сущности по умолчанию Event в реализации `MasterViewController.m` использовалась новая сущность `PhoneBooth`. Отредактируйте файл и замените ссылки "timestamp" на "name", импортируйте файл `Phonebooth.h`, замените в методе `didSelectRowAtIndexPath` класс `NSManagedObject` на `Phonebooth`, а в вызов `setValue` для данных включите `@ "NewPhoneBooth"`.

```
#import "MasterViewController.h"
#import "DetailViewController.h"
#import "PhoneBooth.h"
```

```
// Обычно следует использовать методы доступа, но в данном случае
// синтаксис KVC позволяет обойтись без включения лишнего класса.
[newManagedObject setValue:[NSDate date] @ "NewPhoneBooth" forKey:@"timeStamp-
@"name"];
```

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)
indexPath
{
    NSManagedObject PhoneBooth *object = [[self fetchedResultsController]
objectAtIndex:indexPath];
    self.detailViewController.detailItem = object;
}
```

```
// Изменить ключ сортировки по обстоятельствам.
NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"timeStamp-
name" ascending:NO];
NSArray *sortDescriptors = @[sortDescriptor];
```

```
- (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath *)indexPath
{
    NSManagedObject *object = [self.fetchedResultsController objectAtIndex:indexPath];
    cell.textLabel.text = [[object valueForKey:@"timeStamp name"] description];
}
@end
```



MasterViewController.m



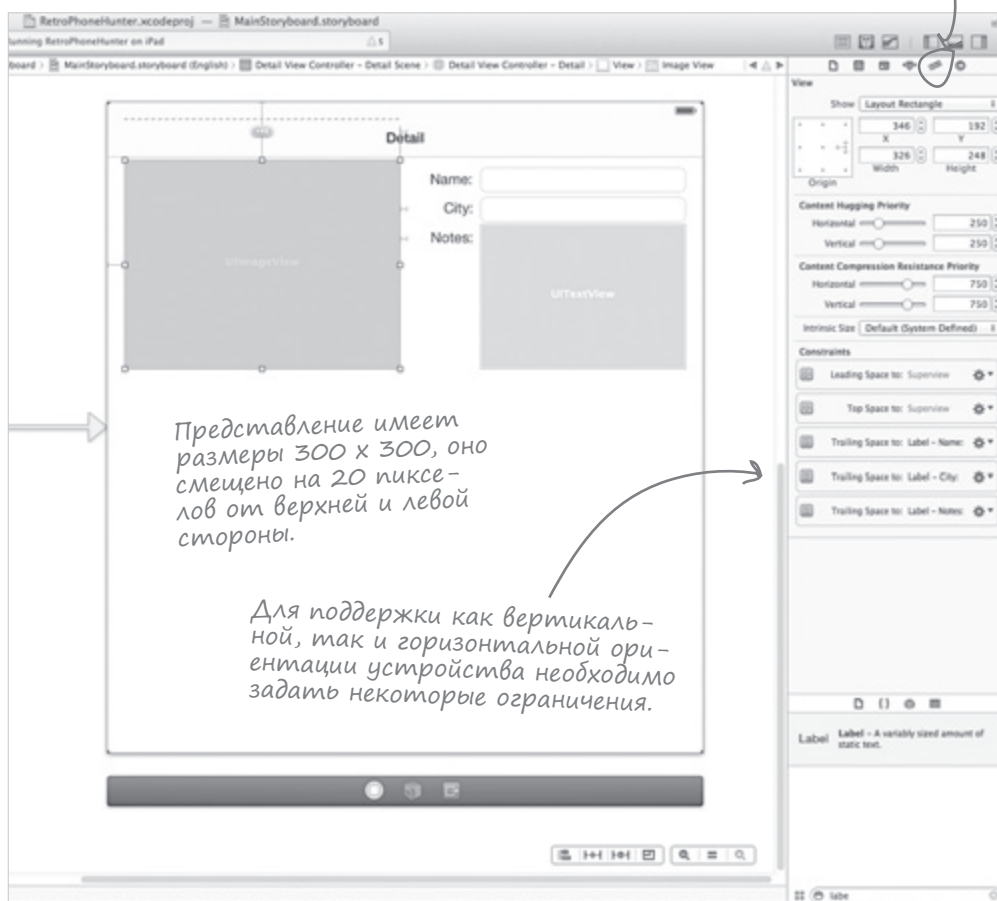
Решение длинных упражнений

4

Постройте макет детализированного представления и добавьте код, обеспечивающий его работу.

Сначала в файл раскладки следует добавить необходимые элементы: UIImageView, надпись **Name** и текстовое поле, надпись **City** и текстовое поле, надпись **Notes** и текстовое поле для заметок.

Чтобы перейти к ограничениям, перейдите в режим инспектора размеров.





```
#import <UIKit/UIKit.h>
#import "PhoneBooth.h"

@interface DetailViewController : UIViewController <UISplitViewControllerDelegate,
UITextViewDelegate>

@property (strong, nonatomic) PhoneBooth * detailItem;
@property (weak, nonatomic) IBOutlet UIImageView *imageView;
@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *cityField;
@property (weak, nonatomic) IBOutlet UITextView *notesView;
- (IBAction)nameFieldEditingChanged:(id) sender;
- (IBAction)cityFieldEditingChanged:(id) sender;
@property (weak, nonatomic) IBOutlet UILabel *detailDescriptionLabel;
@end
```



DetailViewController.h

```
- (void)configureView
{
    if (self.detailItem) {
        self.detailDescriptionLabel.text = [[self.detailItem valueForKey:@"timeStamp"]-
description];

        self.nameField.text = self.detailItem.name;
        self.cityField.text = self.detailItem.city;
        self.notesView.text = self.detailItem.notes;
        self.imageView.image = [UIImage imageWithContentsOfFile:self.
detailItem.imagePath];
    }
}
```



DetailViewController.m



Решение длинных упражнений

```
-(IBAction)nameFieldEditingChanged:(id) sender
{
    self.detailItem.name = self.nameField.text;
}

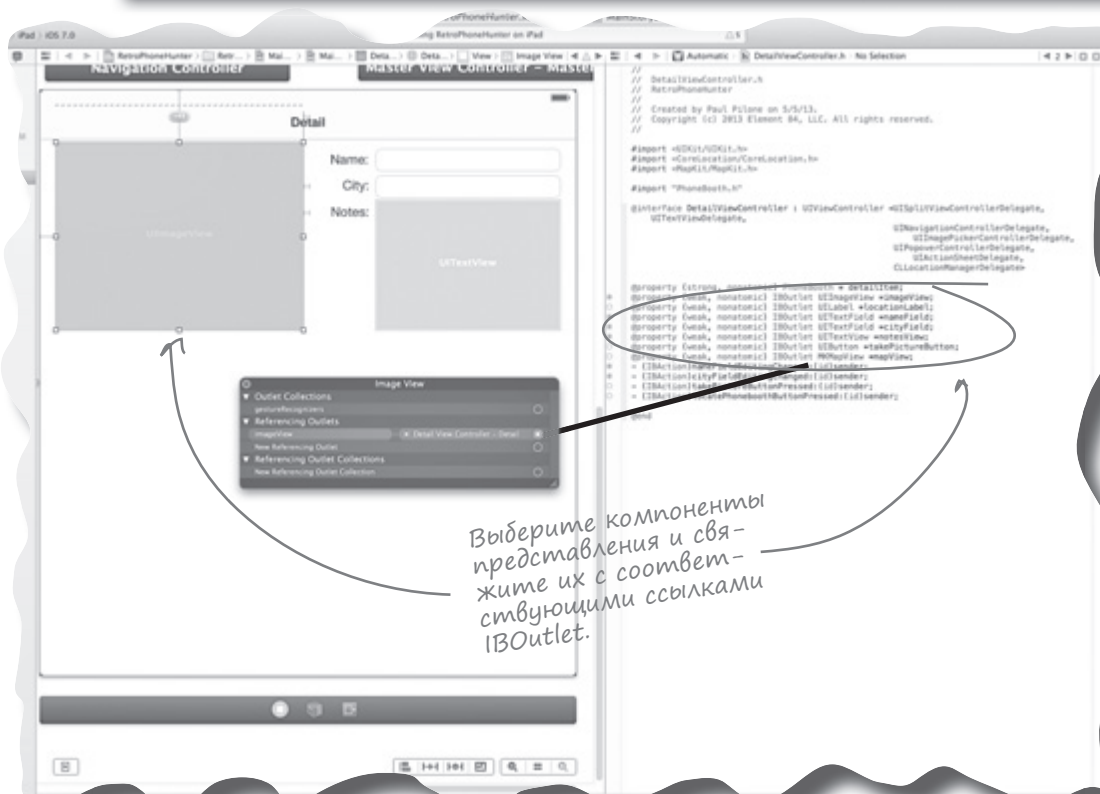
-(IBAction)cityFieldEditingChanged:(id) sender
{
    self.detailItem.city = self.cityField.text;
}

-(void) textViewDidChange:(UITextView *) textView
{
    self.detailItem.notes = self.notesView.text;
}

@end
```



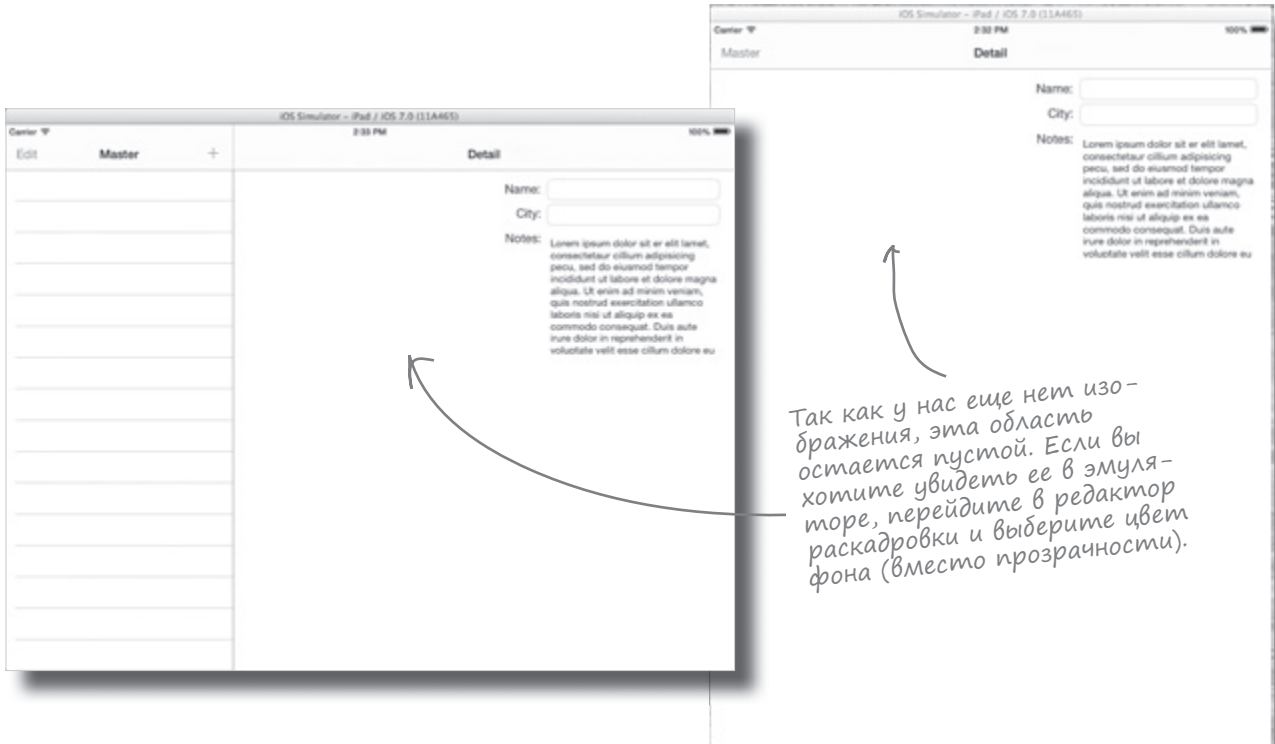
DetailViewController.m





ТЕСТ-АРАЙВ

Запустите приложение и проверьте, как оно работает! Приложение должно запуститься в эмуляторе iPad (впервые в этой книге!) в книжной ориентации. Чтобы увидеть его в альбомной ориентации, выполните команду Hardware ⇨ Rotate Left или Rotate Right.



Возьми в руку карандаш



Теперь, когда все представления работают, как насчет самого изображения? Заполняя пропуски в следующих утверждениях, вспомните модель данных.

Объект UIImage будет храниться в

Объект должен знать об изображении и о том, где оно выводится.

Изображение берется с или из



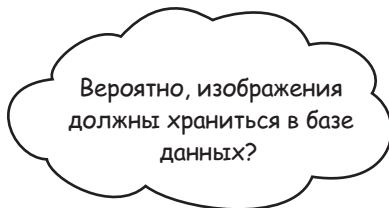
Возьми в руку карандаш Решение

Наконец-то мы добрались до получения изображения.

Объект UIImage будет храниться в *каталоге documents*

Объект *DetailViewController* должен знать об изображении и о том, где оно выводится.

Изображение берется с *камеры* или из *каталога documents*



Базы данных занимают место.

Новые модели iPhone и iPad делают очень качественные снимки... а это обычно означает «очень большие снимки». Попытки сохранить огромное изображение в базе данных быстро приведет к значительным затратам памяти.

Но ведь изображение уже находится на устройстве... снятое на камеру или хранящееся в фотогалерее пользователя. Вместо того чтобы копировать его в базу данных, проще обратиться к существующему изображению на телефоне.

Но *где* хранится это изображение? Как выглядит путь к нему? И куда записать изображение после того, как вы его получили?

Приложения iOS предназначены только для чтения (в какой-то степени...)

Так как мы не будем хранить изображения в базе данных, необходимо поближе познакомиться с файловой системой iOS. И тогда мы сможем просто записать данные в файловую систему, будь то сам файл изображения или путь, соответствующий другому месту устройства... Так?

В какой-то степени. Приложения устанавливаются на устройства iOS с доступом только для чтения. Вы можете обращаться к ресурсам, упакованным с приложением, но не можете изменять их. Единственный способ изменить данные в файловой системе — записать их в одно из специальных мест, которые вам предоставляет iOS... Специально для этой цели.

Например, шаблоны Core Data автоматически делают это за вас. Ниже приведен короткий фрагмент кода, который создает новую базу данных в одном из таких мест, доступных для записи:

Apple называет структуру каталогов, в которую записываются приложения (как части, доступные только для чтения, так и части, доступные для записи), «песочницей» (app sandbox).

```
NSURL *storeUrl = [NSURL fileURLWithPath: [[self
applicationDocumentsDirectory] URLByAppendingPathComponent:
@"retroPhoneHunter.sqlite"]];
```



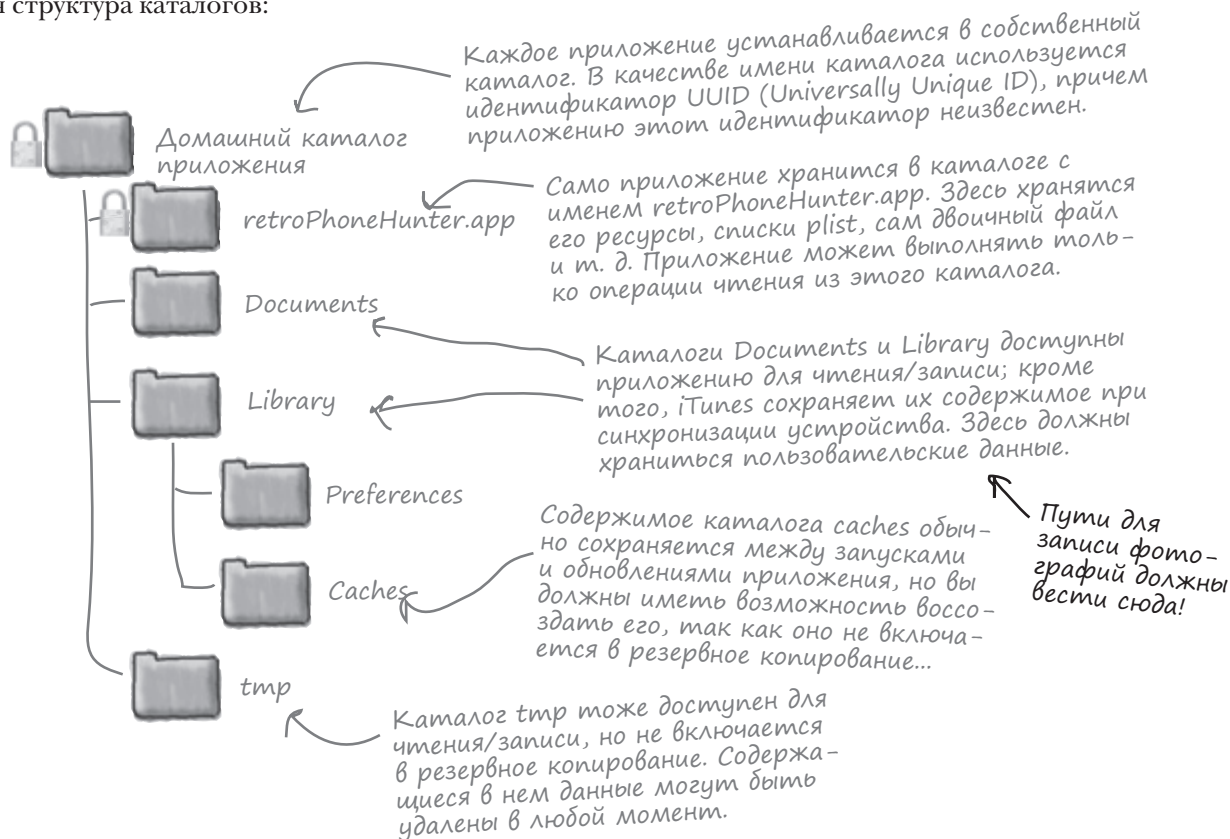
retroPhoneHunterAppDelegate.m

Шаблон Core Data ищет базу данных в каталоге Documents приложения, а не в самом пакете приложения. Дело в том, что этот раздел файловой системы доступен для записи, а пакет приложения недоступен.

Мы должны сделать нечто похожее: определить, куда можно записать данные, получить путь к изображению телефонной будки, которое выбрал пользователь, и выполнить запись по этому пути файловой системы.

Структура приложения iOS определяет, где можно читать и записывать данные

По соображениям безопасности и стабильности работы iOS достаточно жестко блокирует доступ к файловой системе. При установке приложения iOS создает в каталоге `/User/Applications` подкаталог вашего приложения, используя в качестве имени уникальный идентификатор. Затем приложение устанавливается в указанный каталог, и для приложения создается стандартная структура каталогов:



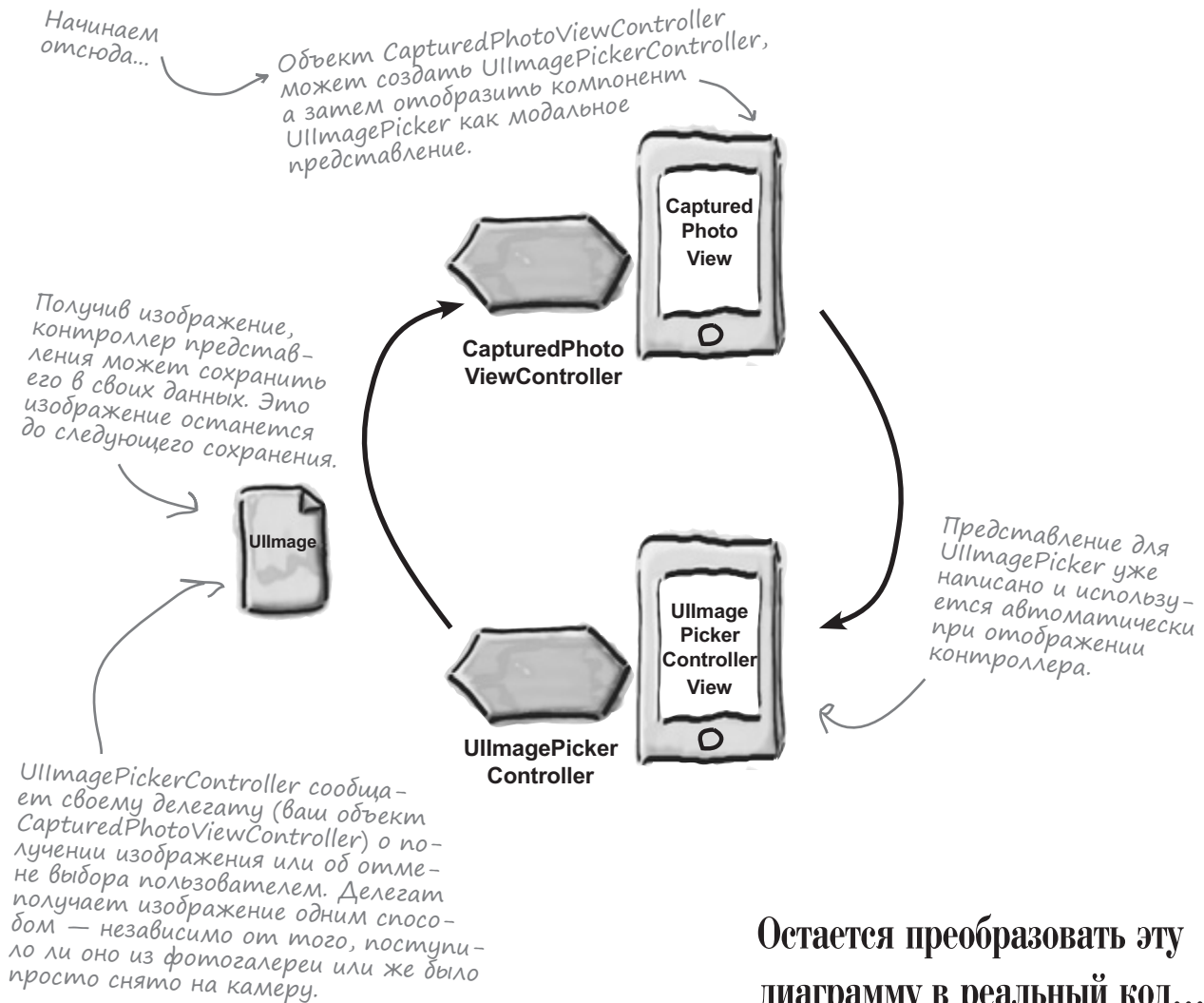
Получите путь к фотографиям, а затем выполняйте запись по этому пути файловой системы!

Теперь вы знаете, что делать... и даже куда следует записывать фотографии: в каталоги `Documents` и `Library`. Но как получить сам путь? К счастью, в iOS это делается намного проще, чем может показаться...

Знакомьтесь... UIImagePickerController

На самом деле мы хотим *выбрать* изображение (после того, как оно будет снято на камеру, или загруженное из фотогалереи), а затем сделать что-то с ним... или по крайней мере с путем, по которому оно хранится. В iOS для выбора изображений используется компонент UIImagePickerController, который позволяет получить изображение из разных мест, например из камеры или фотогалереи.

Что еще лучше, все взаимодействие с пользователем можно поручить компоненту UIImagePickerControllerController. Он позволяет пользователю сделать фотографию или выбрать существующее изображение и передает вам... путь к этому изображению. То что нужно!





Готово к употреблению

Ниже приведена часть кода, необходимого для связывания компонента выбора изображения с приложением. Этот код будет включен в файл *DetailViewController.m* в ходе выполнения упражнения на следующей странице.

```
#pragma mark UIImagePickerControllerDelegate methods

- (void)imagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    // Построить путь к файлу в каталоге Documents.
    NSString *documentsDirectory = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) lastObject];
    NSString *uniqueFilename = [[NSUUID UUID] UUIDString];
    NSString *imagePath = [documentsDirectory stringByAppendingPathComponent:uniqueFilename];

    // Получить изображение от компонента и записать его на диск.
    UIImage *image = [info objectForKey:UIImagePickerControllerEditedImage];
    [UIImagePNGRepresentation(image) writeToFile:imagePath atomically:YES];

    // Сохранить путь к изображению в модели для последующей загрузки.
    self.detailItem.imagePath = imagePath;

    // Обновить представление.
    self.imageView.image = image;

    // Освободить компонент выбора.
    [self dismissViewControllerAnimated:YES completion:nil];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    // Освободить компонент выбора.
    [self dismissViewControllerAnimated:YES completion:nil];
}
```

↑ Закрывать интерфейс
выбора и освободить
объект.



```

- (IBAction)takePictureButtonPressed: (id) sender
{
    NSLog(@"Taking a picture...");
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    picker.sourceType = UIImagePickerControllerSourceTypeCamera | UIImagePickerControllerSourceTypePhotoLibrary;
    picker.allowsEditing = YES;
    picker.delegate = self;

    [self presentViewController:picker animated:YES completion:nil];
}

```



DetailViewController.m



Упражнение

Пора заняться получением изображений! Используя код с предыдущей страницы, а также свои познания в Objective-C, организуйте выбор изображения.

- 1 Добавьте кнопку «Take Picture».**
 Отредактируйте раскладку и создайте кнопку, которая накрывает весь компонент UIImageView, а затем размещается позади него. Не забудьте связать ее с действием takePictureButton. Кнопка должна находиться точно позади UIImageView, а ее ширина должна совпадать с шириной изображения.
- 2 Добавьте код UIImagePickerController в файл DetailViewController.m.**
 Теперь вы можете использовать заготовку с предыдущей страницы для подключения UIImagePickerController к приложению. Чтобы использовать объект DetailViewController в качестве делегата, необходимо указать, что он реализует протоколы UIImagePickerControllerDelegate и UINavigationControllerDelegate.
- 3 Добавьте код действия takePictureButtonPressed.**
 Добавьте в файл *DetailViewController.m* заготовку кода с предыдущей страницы для действия takePictureButtonPressed.
- 4 Упакуйте все компоненты в поповер.**
 Компонент UIImagePickerController должен отображаться на экране как поповер (всплывающее меню). Добавьте свойство для UIPopoverController, убедитесь в том, что *DetailViewController.h* поддерживает UIPopoverControllerDelegate, и реализуйте popoverControllerDidDismissPopover в *DetailViewController.m*.



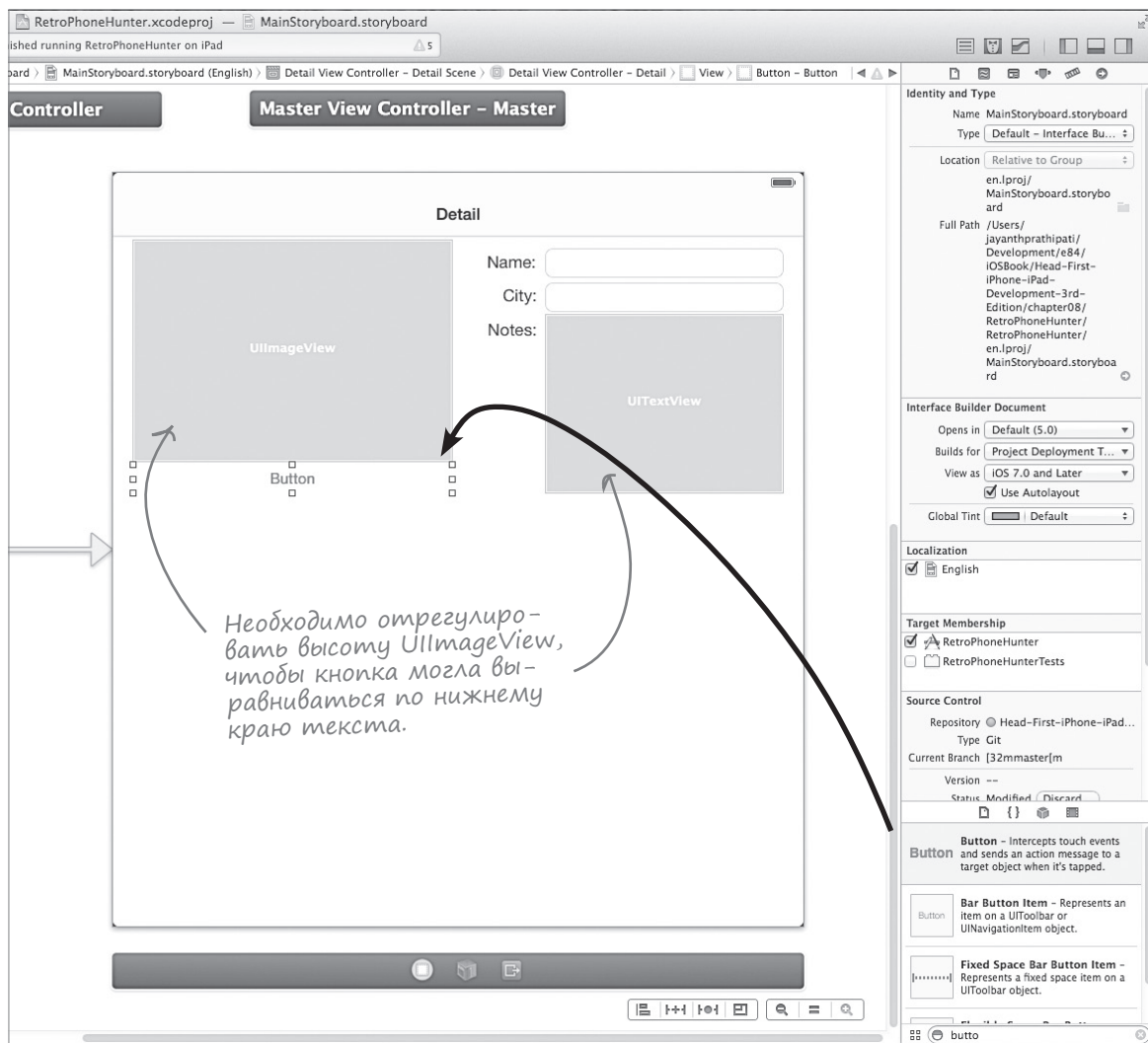
Упражнение
Решение

Наладить работу ImagePicker будет непросто, но как же приятно, когда на экране появляется изображение с камеры или из фотогалереи! Проследите за тем, чтобы ваш код не отличался от кода решения, приведенного ниже.

1

Добавьте кнопку «Take Picture».

Отредактируйте раскладку и создайте кнопку, которая накрывает весь компонент UIImageView, а затем размещается позади него. Не забудьте связать ее с действием takePictureButton. Кнопка должна находиться точно позади UIImageView, а ее ширина должна совпадать с шириной изображения.



2

Добавьте код UIImagePickerControllerController в файл DetailViewController.m.

Теперь вы можете использовать заготовку с предыдущей страницы для подключения UIImagePickerControllerController к приложению. Чтобы использовать объект DetailViewController в качестве делегата, необходимо указать, что он реализует протоколы UIImagePickerControllerDelegate и UINavigationControllerDelegate.

```
@interface DetailViewController : UIViewController <UISplitViewControllerDelegate,
UITextViewDelegate, UINavigationControllerDelegate,
UIImagePickerControllerDelegate>
```



DetailViewController.h

3

Добавьте код действия takePictureButtonPressed.

Добавьте в файл *DetailViewController.m* заготовку кода с предыдущей страницы для действия takePictureButtonPressed.

Сводится к простому копированию кода.

4

Упакуйте все компоненты в поповер.

Компонент UIImagePickerController должен отображаться на экране как поповер (всплывающее меню). Добавьте свойство для UIPopoverController, убедитесь в том, что *DetailViewController.h* поддерживает UIPopoverControllerDelegate, и реализуйте popoverControllerDidDismissPopover в *DetailViewController.m*.

Постойте. А это еще зачем? Почему нужен какой-то поповер?

Это один из постулатов Apple.

Для соответствия стандартам Apple компонент выбора изображения должен размещаться в поповере. При нарушении этого правила выдается исключение (при попытке открыть фотогалерею).





Упражнение
Решение

После продолжительной работы с Objective-C остается внести завершающие штрихи в редакторе раскадровки. Приложение почти готово к запуску!

4 Упакуйте все компоненты в поповер (продолжение).

```
@interface DetailViewController ()
@property (strong, nonatomic) UIPopoverController *masterPopoverController;
@property (strong, nonatomic) UIPopoverController
*imagePickerPopoverController;

- (void)configureView;
@end
```



DetailViewController.m

```
#pragma mark -
#pragma mark UIPopoverControllerDelegate methods

- (void)popoverControllerDidDismissPopover: (UIPopoverController *)
popoverController
{
    self.imagePickerPopoverController = nil;
}

@end
```



DetailViewController.m

```
@interface DetailViewController : UIViewController <UISplitViewControllerDelegate,
UITextViewDelegate,
UINavigationControllerDelegate, UIImagePickerControllerDelegate,
UIPopoverControllerDelegate>
```

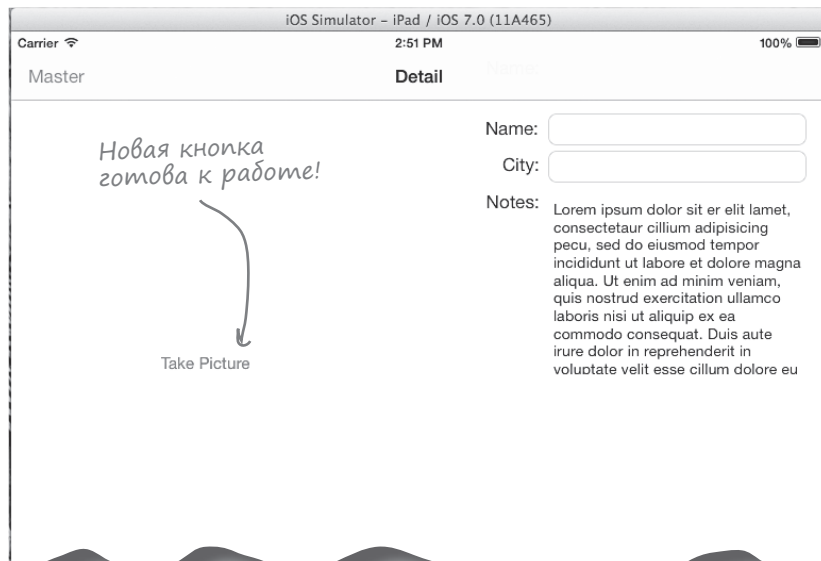


DetailViewController.h



ТЕСТ-ДРАЙВ

Запустите приложение и опробуйте его в деле! Новая кнопка должна находиться на своем месте. Но если вы попытаетесь воспользоваться ею и нажмете...Так-так...



КЛЮЧЕВЫЕ МОМЕНТЫ



И что же произойдет, когда пользователь нажмет кнопку «Take Picture»? Приложение проверяет наличие камеры... и что потом? Что именно должен увидеть пользователь? А дальше? И что после этого?

Использование карты действий

Карта действий (action sheets) «выдвигается» на экран от нижнего края представления; на ней перечислены варианты дальнейших действий пользователя. Как и модальное представление, карта действий вынуждает пользователя выполнить одно из действий для продолжения работы. Карты действий очень просты в использовании: они получают строки, определяющие надписи на их кнопках, а их открытие и закрытие сопровождается встроенной анимацией.

Следующий код используется для карты действий, которая предлагает пользователю сделать фотографию или выбрать изображение в фотогалерее:

```
UIAlertSheet *photoSourceSheet = [[UIAlertSheet alloc]
initWithTitle:@"Select PhoneBooth Picture"
```

```
delegate:self
```

```
cancelButtonTitle:nil
```

```
destructiveButtonTitle:nil
```

```
otherButtonTitles:@"Take New Photo",
```

```
@"Choose Existing Photo", nil];
```

В картах действий часто присутствует кнопка типа «Да, я знаю, что все мои данные будут удалены. Выполняйте». Эта кнопка называется деструктивной. Впрочем, в данном случае она не нужна.

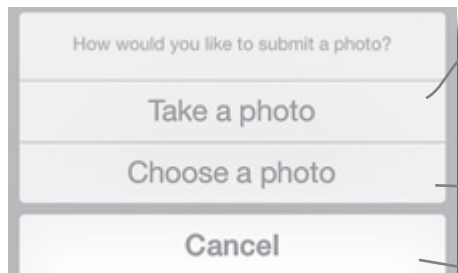
Во всех картах действий должна присутствовать кнопка отмены.

Объявляем две другие кнопки для карты — и на этом работа закончена.

Создаем экземпляр карты действий и передаем текст заголовка.

Карты действий ведут... к действиям

Карта содержит два действия: «Сделать новый снимок» и «Выбрать существующее изображение». Причем вполне очевидно, что должно происходить в каждом случае:



Обратиться к камере, сделать фотографию, а затем вернуться и включить новое изображение в Phonebooth. В нашем случае достаточно передать управление компоненту UIImagePickerControllerSourceTypeCamera, а он сделает все остальное.

Перейти к фотогалерее, выбрать изображение, затем вернуться и поместить его в Phonebooth. Здесь всю работу выполняет компонент UIImagePickerControllerSourceTypePhotoLibrary.

Вернуться к представлению с фотографией.

Возьми в руку карандаш



Переходим к реализации карты действий. Здесь придется хорошенько подумать, поскольку мы слегка изменяем нормальную последовательность выполнения приложения... Не торопитесь, и у вас все получится.

1

Реализуйте методы делегата для карты действий.

Следующий код станет отправной точкой для дальнейшей работы. Продумайте разные варианты — включая вариант по умолчанию!

Не забудьте включить в заголовочный файл объявление о реализации `UIActionSheetProtocol`!

```
- (void)actionSheet:(UIActionSheet *)actionSheet didDismissWithButtonI
ndex:(NSInteger)buttonIndex
{
    if (buttonIndex == actionSheet.cancelButtonIndex) {
        NSLog(@"The user canceled adding a image.");
        return;
    }

    UIImagePickerController *picker = [[UIImagePickerController alloc]
init];
    picker.delegate = self;
    picker.allowsEditing = YES;
    switch (buttonIndex) {
        case 0:
            NSLog(@"User wants to take a new picture.");
            picker.sourceType = UIImagePickerControllerSourceTypeCamera;
            break;
```

2

Измените действие `takePictureButtonPressed` в файле `CapturePhotoViewController.m` и включите в него карту действий.

Приложение `retroPhoneHunter` сначала должно проверить наличие камеры. Если камера имеется, пользователь должен выбрать между получением снимка с камеры и существующим изображением. Если камеры нет, приложение должно перейти прямо к фотогалерее.

Карта действий предлагает пользователю выбрать одну из перечисленных операций.

3

Добавьте ссылку `IBOutlet` для `takePictureButton`.

Это необходимо, потому что из кода кнопки будет отображаться контроллер поповера. Ранее мы использовали параметр `'sender'` метода `takePictureButtonPressed`, но в методе делегата карты действий такой возможности нет. Поэтому нам и понадобится новая ссылка!




Возьми в руку карандаш Решение

Дальше идет довольно большой объем кода... Не торопитесь и убедитесь в том, что все было сделано правильно!

1 Реализуйте методы делегата для карты действий.

```
@interface DetailViewController : UIViewController
<UISplitViewControllerDelegate, UITextViewDelegate,
UINavigationControllerDelegate, UIImagePickerControllerDelegate,
UIPopoverControllerDelegate, UIActionSheetDelegate>
```



DetailViewController.h

```
- (void)actionSheet:(UIActionSheet *)actionSheet didDismissWithButton
Index:(NSInteger)buttonIndex
{
    if (buttonIndex == actionSheet.cancelButtonIndex) {
        NSLog(@"The user canceled adding a image.");
        return;
    }

    UIImagePickerController *picker = [[UIImagePickerController alloc]
init];
    picker.delegate = self;
    picker.allowsEditing = YES;

    switch (buttonIndex) {
        case 0:
            NSLog(@"User wants to take a new picture.");
            picker.sourceType = UIImagePickerControllerSourceTypeCamera;
            break;
        default:
            picker.sourceType =
UIImagePickerControllerSourceTypePhotoLibrary;
            break;
    }
}
```



DetailViewController.m

```

    self.imagePickerPopoverController = [[UIPopoverController alloc] initWithContentViewController:picker];
    self.imagePickerPopoverController.delegate = self;
    [self.imagePickerPopoverController presentPopoverFromRect:self.takePictureButton.frame
    inView:self.view
    permittedArrowDirections:UIPopoverArrowDirectionLeft
    animated:YES];
}

```



DetailViewController.m

2

Измените действие `takePictureButtonPressed` в файле `CapturePhotoViewController.m` и включите в него карту действий.

```

- (IBAction)takePictureButtonPressed:(id)sender
{
    NSLog(@"Taking a picture...");
    if ([UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {
        NSLog(@"This device has a camera. Asking the user what they want to use.");
        UIAlertController *photoSourceSheet = [[UIAlertSheet alloc] initWithTitle:@"Select PhoneBooth Picture"
        delegate:self
        cancelButtonTitle:nil
        destructiveButtonTitle:nil
        otherButtonTitles:@"Take New Photo",
        @"Choose Existing Photo", nil];
        // Вывести карту действий рядом с кнопкой добавления изображения.
        [photoSourceSheet showFromRect:(UIButton *)sender.frame inView:self.view animated:YES];
    }
}

```



DetailViewController.m

Продолжаем...



Возьми в руку карандаш

Решение

2

Измените действие `takePictureButtonPressed` в файле `CapturePhotoViewController.m` и включите в него карту действий (продолжение).

```
else { // Камеры нет, использовать галерею.
    UIImagePickerController *picker = [[UIImagePickerController alloc]
init];
    picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    picker.allowsEditing = YES;
    picker.delegate = self;

    self.imagePickerPopoverController = [[UIPopoverController alloc] ini
tWithContentViewController:picker];
    self.imagePickerPopoverController.delegate = self;
    [self.imagePickerPopoverController presentPopoverFromRect:((UIButton
*)sender).frame inView:self.view permittedArrowDirections:UIPopoverArrow
DirectionLeft animated:YES];
}
}
```



DetailViewController.m

3

Добавьте ссылку `IBOutlet` для `takePictureButton`.

```
@property (weak, nonatomic) IBOutlet UITextView *notesView;
@property (weak, nonatomic) IBOutlet UIButton *takePictureButton;
- (IBAction)nameFieldEditingChanged:(id) sender;
```

Не забудьте создать необходимые
связи в раскадровке!



ТЕСТ-ДРАЙВ

Запустите приложение retroPhoneHunter и добавьте телефонную будку. Опробуйте новый код: выберите существующее изображение (или сделайте новый снимок). Если вы использовали SourceTypePhotoLibrary в коде takePictureButtonPressedcode, все должно работать, и вы увидите карту действий на экране.



Часто Задаваемые Вопросы

В: Но ведь новые модели iPhone и iPads поддерживают видео? Как использовать эту возможность?

О: Это другой тип данных, к которому вы получаете доступ при использовании UIImagePickerController. По умолчанию используются статические изображения, как нам и нужно в приложении retroPhoneHunter.

В: А как насчет новомодной концепции «дополненной реальности» при работе с камерой? Я могу сделать что-нибудь подобное?

О: Да. Вы можете передать UIImagePickerController пользовательское оверлейное представление, которое будет использоваться при работе с камерой. Возможности работы с представлением камеры неограничены, но при желании вы можете наложить на фотографию свою информацию.

В: Что это за режим allowEditing, который мы включили для UIImagePickerController?

О: В элементе выбора реализована встроенная поддержка обрезки и масштабирования изображений. Флаг allowEditing определяет, получит ли пользователь возможность применить эти операции до отправки изображения делегату. Если при включенном режиме пользователь изменит изображение, информация редактирования будет доступна в методе обратного вызова.

В: Нам действительно придется позаботиться об устройствах, не оснащенных камерой?

О: Безусловно! При отправке приложения в Apple для распространения через iTunes App Store вы указываете, с какими устройствами работает приложение. Если вы указали, что приложение работает на обоих видах устройств, то Apple протестирует его на обоих видах. Также выполняются тесты, в которых ваше приложение не может получить доступ к сети, — они проверяют, что приложение справится и с этой ситуацией. Постарайтесь продумать все возможные варианты! Apple стремится протестировать ваше приложение в разнообразных сценариях.

В: Можно ли протестировать камеру в эмуляторе?

О: Нет. В тест-драйве на предыдущей странице мы сделали практически все возможное, когда реализовали код работы с камерой и протестировали его с фотогалереей. Вы уже узнали много нового, и большая часть функциональности, которой мы займемся в дальнейшем, превышает возможности эмулятора. Функциональность GPS, акселерометр, динамик — все эти возможности нельзя протестировать в эмуляторе. Тестирование возможно только при установке приложения на реальном устройстве.

В: Напомните, что вы говорили о программе Apple Developer Program?

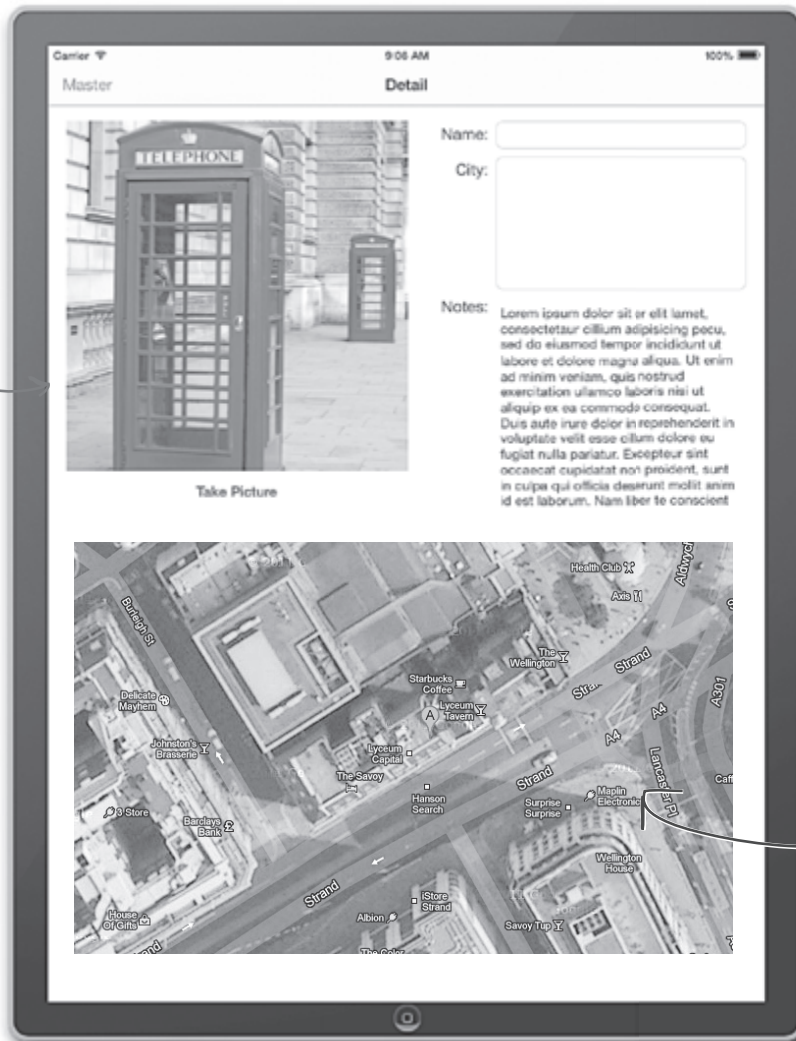
О: Чтобы установить приложение на устройство или отправить его в App Store, необходимо зарегистрироваться у Apple в качестве разработчика iOS. Стоимость в настоящее время составляет \$99. Даже если вы хотите просто установить приложение для собственного использования, регистрация все равно необходима.



Для Любопытных

Возможно, пришло время зарегистрироваться в программе Apple's Developer Program. После регистрации вы сможете установить приложение retroPhoneHunter на свой планшет iPad и протестировать его.

Итак, теперь
эта функция
работает...
с камерой или
фотогалереей...



И здесь тоже все
просто. Добавить
в приложение не-
сколько свойств...

...Но что делать
с картами? Это
уже серьезно,
не так ли?

Где эта улица, где этот дом?

Вы поняли, да? Следу-
ющий вопрос в нашей
программе — «где»...

Карта важна, потому что по ней пользователь узнает, где находятся телефонные будки. На карте должны выводиться два вида дополнительной информации: во-первых, координаты будки (если мы их знаем), а во-вторых, координаты самого пользователя, чтобы карта отображалась относительно его текущей позиции. А когда пользователь делает снимок, информация о местоположении будки сохраняется вместе с ним. Добавить представление карты несложно; заставить работать всю эту схему чуть сложнее.

А теперь настоящий вопрос: как устройству iOS получить жизненно важную информацию о местоположении?

Core Location может определить текущее местоположение несколькими способами

GPS — первое, о чем обычно думают пользователи, но iPhone первого поколения не оснащались модулем GPS; нет его и на современных устройствах iPod Touch или WiFi-моделях iPad. Впрочем, это не означает, что вы оказались в безвыходном положении. iOS может определить текущее положение устройства, используя три механизма: GPS, триангуляцию вышек сотовой связи и средства позиционирования WiFi.

GPS выдает самые точные данные, далее идет триангуляция и WiFi. iPhone могут использовать два или три из перечисленных способов, тогда как для iPod Touch и WiFi-моделей iPad доступен только WiFi... впрочем, это лучше, чем ничего. Если у вас голова идет кругом, не огорчайтесь! Core Location выбирает способ позиционирования в зависимости от возможностей устройства и требуемой точности. Вам почти ничего делать не придется; iOS сделает все за вас при помощи объекта **CLLocationManager**:

```
- (CLLocationManager*) locationManager {
    if (locationManager_ == nil) {
        locationManager_ = [[CLLocationManager alloc] init];
        locationManager_.desiredAccuracy = kCLLocationAccuracyNearestTenMeters;
        locationManager_.delegate = self;
    }
    return locationManager_;
}
```

Создание объекта CLLocationManager.

Установит точность равной 10 метрам. Позиционирование с более высокой точностью занимает больше времени и может заметно снизить производительность вашего приложения.

Когда объект locationManager определит текущую позицию, он начинает отправлять ее делегату.

Core Location используем LocationManager

Чтобы использовать Core Location, достаточно создать объект locationManager и приказать ему начать отправку обновлений. Он может предоставлять данные о позиции, высоте и ориентации (в зависимости от возможностей устройства). Чтобы получать эту информацию, необходимо лишь предоставить делегата и указать желательную точность.

Объект CLLocationManager будет оповещать вас о появлении данных позиционирования и об ошибках. Если вашему приложению не удастся получить данные позиционирования, также позаботьтесь о должной обработке таких ситуаций. Даже если устройство поддерживает такую возможность, пользователю сначала предлагается разрешить сбор данных позиционирования, и он может отказаться (случайно или сознательно).



Где следует разместить этот код
в вашем приложении?



Наверное, нам понадобится
новый заголовочный файл
для констант Core Location?

Верно... а еще новая инфраструктура.

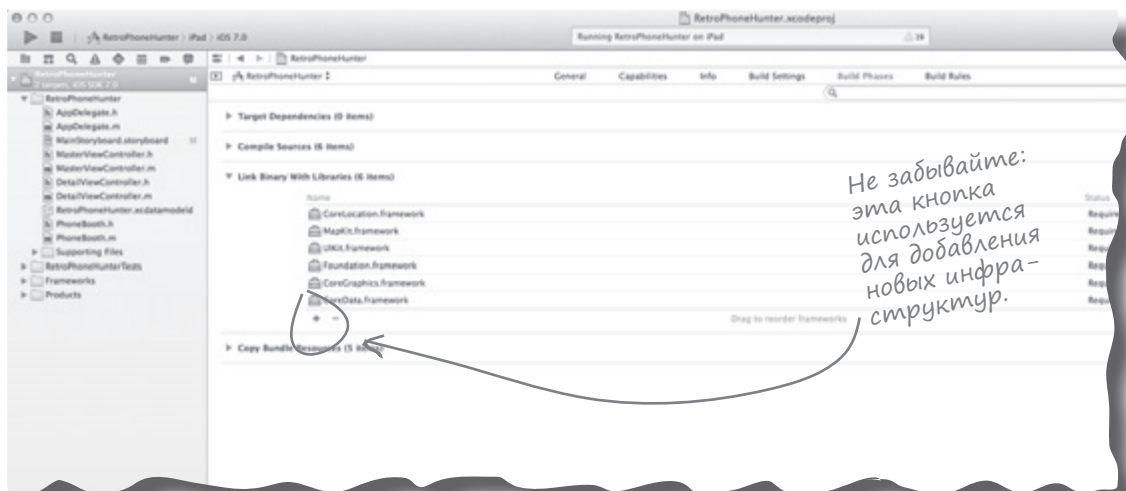
Чтобы по возможности сократить размер приложений, Apple разбивает разные виды функциональности на инфраструктуры. Когда в приложение добавляется новая функциональность (например, Core Location), нужно добавить в проект новые инфраструктуры.

Так как инфраструктура Core Location не включается по умолчанию, необходимо добавить ее явно.



Добавление инфраструктуры Core Location.

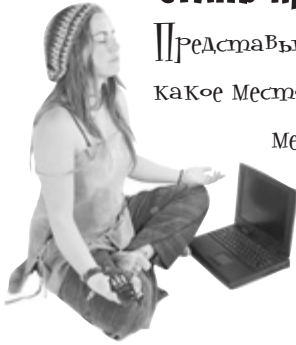
Выберите проект в навигаторе, затем выберите цель RetroPhoneHunter и вкладку Build Phases. Добавьте в список пункт CoreLocation Framework.



Core Location быстро разряжает батарею.

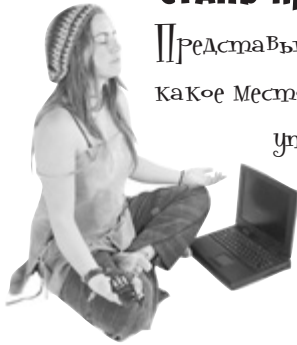
Частые вызовы API от приложения, проверяющего свое местоположение, приводят к быстрой разрядке батареи, потому что они требуют включения приемника GPS/сотовой связи/WiFi. Это раздражает пользователей и становится причиной отрицательных отзывов в iTunes. Сведите количество таких вызовов к минимуму!

СТАНЬ приложением



Представьте себя на месте приложения и вычислите, какое место занимает Core Location при передаче управления между представлениями приложения. Будем считать, что для новой найденной телефонной будки необходимо сохранить местоположение, дату и время.

- 1 Какой метод будет использоваться для активизации Core Location при создании снимка?
.....
- 2 Что произойдет при возвращении данных позиционирования контроллеру представления?
.....
.....
- 3 Что произойдет, если Core Location не может получить данные позиционирования, или пользователь запретит это делать?
.....
.....
- 4 Как насчет других устройств? Что должно происходить с ними?
.....
.....



СТАНЬ приложением. Решение

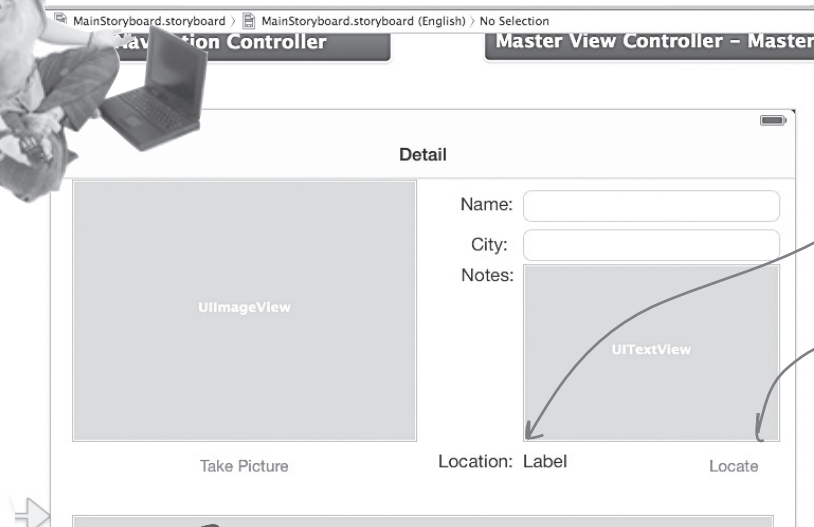
Представьте себя на Месте приложения и Вычислите, какое Место занимает Core Location при передаче управления между представлениями приложения. Будем считать, что для новой найденной телефонной будки необходимо сохранить Местоположение, дату и Время.

1

Какой метод будет использоваться для активизации Core Location при создании снимка?

Код инициализации Core Location будет находиться в методе viewWillAppear детализированного представления на iPhone, но поскольку наше приложение предназначено для iPad, детализированное представление остается видимым постоянно. Чтобы упростить задачу, мы добавим кнопку получения данных позиционирования по запросу пользователя.

СТАНЬ приложением. Решение



Добавьте эти надписи в представление для вывода данных позиционирования

Новая кнопка для активизации кода Core Location.

```
#import <CoreLocation/CoreLocation.h>

@property (strong, nonatomic) PhoneBooth * detailItem;
@property (weak, nonatomic) IBOutlet UIImageView *imageView;
@property (weak, nonatomic) IBOutlet UILabel *locationLabel;
@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *cityField;
@property (weak, nonatomic) IBOutlet UITextView *notesView;
@property (weak, nonatomic) IBOutlet UIButton *takePictureButton;

- (IBAction)nameFieldEditingChanged:(id)sender;
- (IBAction)cityFieldEditingChanged:(id)sender;
- (IBAction)takePictureButtonPressed:(id)sender;
- (IBAction)locatePhoneboothButtonPressed:(id)sender;

@end
```



DetailViewController.h



СТАНЬ приложением. Решение

1

Какой метод будет использоваться для активизации Core Location при создании снимка? (продолжение)

```
- (IBAction)locatePhoneboothButtonPressed: (id) sender
{
    [self.locationManager startUpdatingLocation];
}
```



DetailViewController.m

2

Что произойдет при возвращении данных позиционирования контроллеру представления?

Это будет означать, что объект `LocationManager` может получить текущую позицию. Если пользователь добавляет новую телефонную будку, приложение должно получить текущую позицию от `LocationManager` и обновить информацию.

```
@interface DetailViewController ()
@property (strong, nonatomic) UIPopoverController *masterPopoverController;
@property (strong, nonatomic) UIPopoverController *imagePickerPopoverController;
@property (strong, nonatomic) CLLocationManager *locationManager;

- (void)configureView;
end
```



DetailViewController.m



СТАТЬ приложением. Решение

2

Что произойдет при возвращении данных позиционирования контроллеру представления? (продолжение)

```
- (CLLocationManager *)locationManager
{
    if (!_locationManager) {
        _locationManager = [[CLLocationManager alloc] init];
        _locationManager.desiredAccuracy =
            kCLLocationAccuracyNearestTenMeters;
        _locationManager.delegate = self;
    }

    return _locationManager;
}
```



DetailViewController.m



СТАНЬ приложением. Решение

3

Что произойдет, если Core Location не может получить данные позиционирования или пользователь запретит это делать?

Ситуация неприятная, но данные позиционирования не являются абсолютно необходимыми для нашего приложения. Пользователь вводит город вручную, но не имеет данных позиционирования от Core Location... и не видит карты.

```
#pragma mark -
#pragma mark CLLocationManagerDelegate methods

- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray *)locations
{
    NSLog(@"Core location claims to have a position.");
    CLLocation *location = [locations lastObject];

    // Обновить данные и представление.
    self.detailItem.lat = [NSNumber numberWithDouble:location.coordinate.latitude];
    self.detailItem.lon = [NSNumber numberWithDouble:location.coordinate.longitude];

    [self configureView];

    // Прекратить отслеживание позиции. Вероятно, в реальном приложении
    // обновление должно продолжаться для получения наиболее точной позиции.
    NSLog(@"Shutting down core location.");
    [self.locationManager stopUpdatingLocation];
}

- (void)locationManager:(CLLocationManager *)manager didFailWithError:(NSError *)error
{
    NSLog(@"Core location can't get a fix!");

    // Обновить представление, чтобы сообщить пользователю
    // о неудаче при получении позиции.
    self.locationLabel.text = @"Can't get a location.";
}

@end
```



DetailViewController.m

4

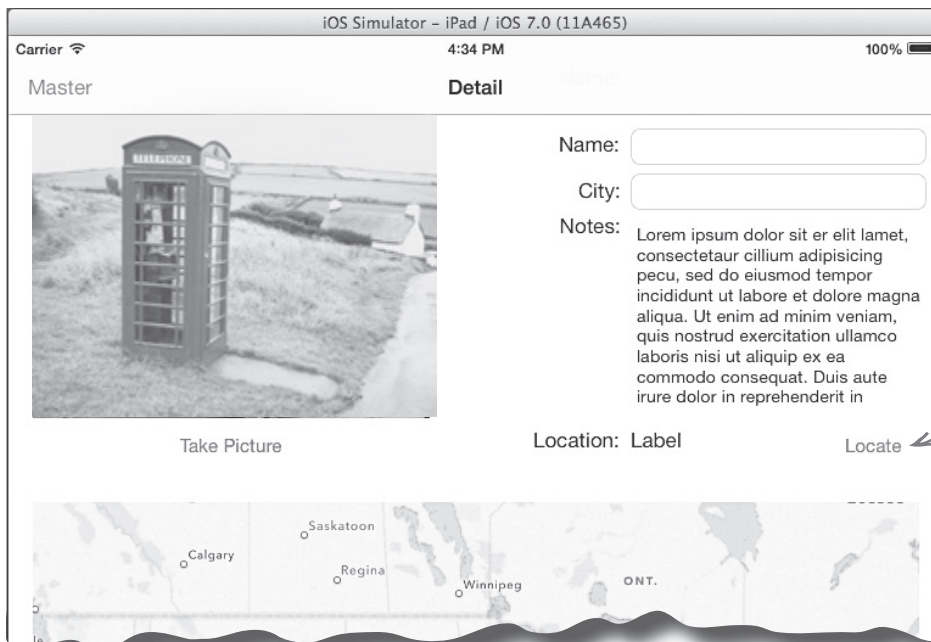
Как насчет других устройств? Что должно происходить с ними?

Все нормально: просто передайте Core Location необходимую точность, а инфраструктура сделает все остальное. Таким образом, iPod Touch получит лучшие данные, какие только может получить, и вернет их приложению. Все просто замечательно!

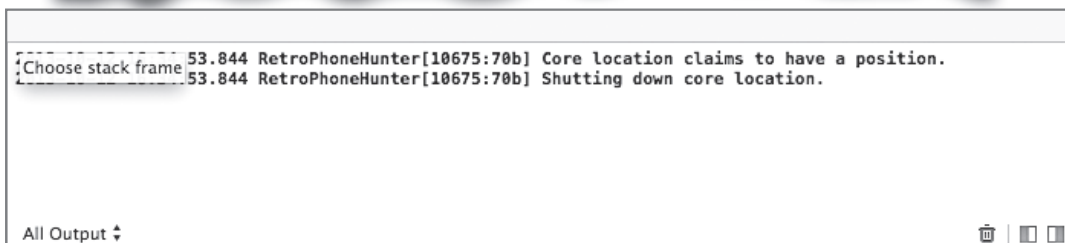


ТЕСТ-ДРАЙВ

Реализовать функциональность Core Location было не так уж сложно, но интегрировать ее в логику работы приложения было чуть сложнее. Теперь, когда все сделано, приложение должно работать...



Нажимаем кнопку «locate», которая пытается определить текущее местоположение (вероятно, со-ответствующее местоположению телефонной будки)...



Часто Задаваемые Вопросы

В: Запуск и остановка Core Location в `viewWillAppear` и `viewWillDisappear` — это нормально?

О: Запускать и останавливать Core Location по мере надобности — нормально. Позиционирование интенсивно расходует батарею, поэтому лучше отключать его, когда оно не используется напрямую. С другой стороны, это может создать некоторые проблемы, потому что Core Location может понадобиться время на получение исходных позиционных данных. Чтобы сделать задержку менее заметной для пользователя, включайте Core Location сразу же при появлении представления на экране; тем самым вы дадите Core Location немного времени перед тем, как данные понадобятся пользователю.

В: Можно ли как-то ускорить получение исходной позиции?

О: Core Location пытается кэшировать предыдущие данные позиционирования, чтобы пользователь как можно быстрее получил хоть какую-то информацию. Если точность для вас критична, проверьте временную метку, передаваемую с данными позиционирования, и убедитесь в том, что полученные данные не устарели.

В: Влияет ли точность данных на время инициализации или расход заряда?

О: Безусловно. Чем выше точность запрашиваемых данных, тем больший заряд потребляет Core Location и тем больше времени займет получение данных. Информация с низкой точностью поступает быстрее. Используйте ту сложность, которая необходима для вашего приложения, но помните о последствиях получения высокоточной информации... и запрашивайте точные данные только тогда, когда это действительно необходимо.

В: Можно ли просто подождать, пока у Core Location появятся данные позиционирования, вместо использования обратного вызова к делегату?

О: К сожалению, нет. Core Location, как и многие другие инфраструктуры iOS, использует механизм асинхронного обратного вызова при появлении данных. Сетевые операции обычно выполняются по тому же принципу. Позаботьтесь о том, чтобы ваш пользователь знал, что происходит в приложении и что он может или не может делать в настоящий момент. Например, можно вывести на экран индикатор ожидания (в виде вращающегося кольца) или обозначения текущего состояния при помощи значка, кнопки или надписи.

В: Почему нам пришлось перемещать этот код и проводить рефакторинг?

О: Для соблюдения принципа DRY (Don't Repeat Yourself, то есть «Не повторяйтесь»). Иначе говоря, речь идет о чистке кода и устранении дублирования за счет перемещения общего кода в отдельный метод и вызова этого метода из двух мест, в которых он используется. Без рефакторинга один код находился бы в двух разных местах приложения.

В: Напомните, для чего нужны закрытые интерфейсы?

О: Вспомните, что в заголовочном файле объявляется открытый интерфейс (API). Но в `refreshPhoneboothInformation` этот внутренний метод не должен быть частью API (другими словами, он не должен вызываться внешними пользователями). Метод нужно объявить так, чтобы компилятор мог проверить правильность вызова метода, но для этого достаточно добавить в интерфейс в файле реализации набор закрытых методов. Некоторые программисты используют в именах закрытых методов префикс `_` (символ подчеркивания), чтобы по имени было сразу видно, что метод не должен вызываться за пределами реализации класса. Впрочем, компания Apple резервирует это обозначение для своих закрытых методов.

Данные позиционирования — это хорошо, но как насчет карты? Нужно воспользоваться другой инфраструктурой.

Map Kit поддерживается всеми устройствами iOS

Когда компания Apple открыла доступ к Map Kit API в iOS 3.0, разработчики получили возможность пользоваться картами Apple, включая визуальные данные со спутников.

Карты поддерживают разнообразную настройку. Разработчик может указать ширину отображаемой области, исходную ширину начального участка и даже добавить маркеры и заметки. Собственно, разработчику доступны все возможности, которые он видит при использовании карт Apple.

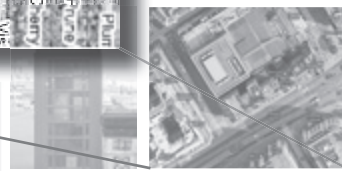
В предыдущих версиях iOS инфраструктура Map Kit использовала карты Google Maps. Начиная с iOS 6 Apple предоставляет собственные карты. Визуальные данные со спутников по-прежнему доступны, но они уже не совпадают с данными, предоставляемыми Google.

В зависимости от того, какая информация должна выводиться на карте, вы можете создать собственное представление для аннотаций и вывести любую информацию по своему усмотрению — графику, отформатированный текст и т. д.

МКMapView — элемент управления для работы с картами Apple. Разработчик может настроить элемент для вывода обычной карты, спутниковой карты или гибридной схемы, как на следующей иллюстрации.



Map Kit предоставляет встроенную поддержку расстановки маркеров в заданных точках (эти маркеры называются «аннотациями»).



Местоположение телефонной будки обозначается на карте рядом с ее фотографией в приложении.



Будьте осторожны!

Для Map Kit необходимо сетевое подключение.

Так как Map Kit получает информацию от Apple, для использования этой функциональности приложение должно располагать сетевым подключением. Эмулятор работает нормально (если Мас подключен к Интернету), но на устройствах с ограниченными возможностями подключения могут возникнуть проблемы в зависимости от местоположения. Map Kit обрабатывает такие ситуации корректно, но знать о них необходимо.

Часто
Задаваемые
Вопросы

В: Чем Core Location отличается от Map Kit?

О: Инфраструктура Map Kit предназначена для вывода карт, позиционной информации и пользовательского интерфейса приложения. Задача Core Location — получение информации о том, где устройство **находится**. Вы можете перетащить карту Map Kit на свое представление в редакторе графического интерфейса, передать нужные значения — и все будет просто работать.

Напротив, Core Location возвращает данные делегату, и вы должны решить, как их обработать. Например, можно получить информацию от Core Location и передать ее Map Kit, чтобы вывести карту с позицией телефонной будки.

В: Откуда берутся все эти инфраструктуры? А если мне понадобится инфраструктура, которой нет в списке?

О: Они поставляются в составе iOS SDK. Непосредственный путь к инфраструктурам зависит от версии и платформы, для которой вы программируете. Например, инфраструктура Map Kit находится в каталоге: `/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS7.0sdk/System/Library/Frameworks/MapKit.framework`. В общем случае разработчик добавляет инфраструктуры в Xcode, не беспокоясь о конкретном каталоге. Но если инфраструктура отсутствует в списке, а также при добавлении сторонней библиотеки вы можете сообщить Xcode фактический путь к инфраструктуре.



Упражнение

Дело близится к развязке! Мы используем инфраструктуру Map Kit для получения координат, полученных от Core Location, и их представления на удобной, красивой карте. За работу!

1**Добавьте инфраструктуру Map Kit и директиву #import.**

Добавьте инфраструктуру Map Kit так же, как вы добавили Core Location. Заодно убедитесь в том, что детализированное представление содержит директиву #import для включения заголовочного файла Map Kit.

2**Настройте детализированное представление для отображения карты.**

Мы оставили в детализированном представлении место для карты. В файле раскадровки перетащите компонент MKMapView в нижнюю половину представления.

3**Добавьте ссылки IBOutlet и код для MKMapView.**

Теперь, когда вся подготовка выполнена, добавьте ссылки IBOutlet и код Map Kit, обеспечивающий работу карт. Не забудьте создать связь для IBOutlet в Xcode.

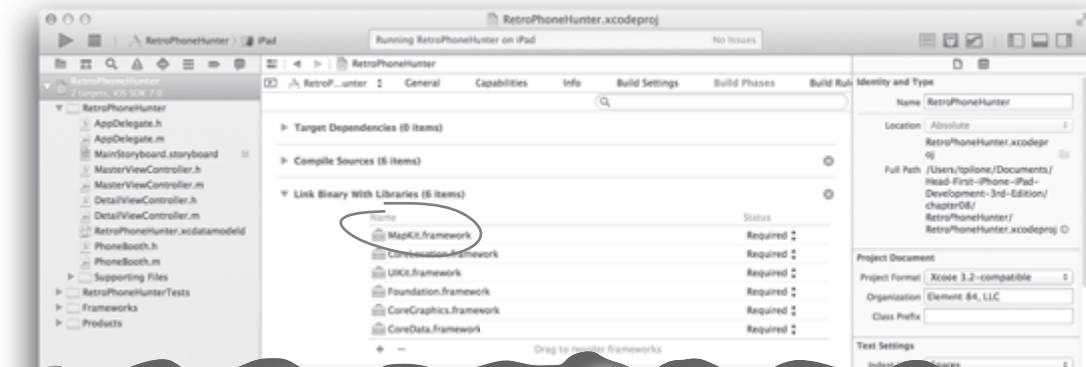


Упражнение
Решение

1

Добавьте инфраструктуру Map Kit и директиву #import.

Добавьте инфраструктуру Map Kit так же, как вы добавили Core Location. Заодно убедитесь в том, что детализированное представление содержит директиву #import для включения заголовочного файла Map Kit.



```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
#import <MapKit/MapKit.h>

#import "PhoneBooth.h"

@interface DetailViewController : UIViewController
<UISplitViewControllerDelegate, UITextViewDelegate,
    UINavigationControllerDelegate,
    UIImagePickerControllerDelegate,
    UIPopoverControllerDelegate,
    UIActionSheetDelegate,
    CLLocationManagerDelegate>

@property (weak, nonatomic) IBOutlet MKMapView *mapView;
@property (strong, nonatomic) PhoneBooth * detailItem;
```



DetailViewController.h



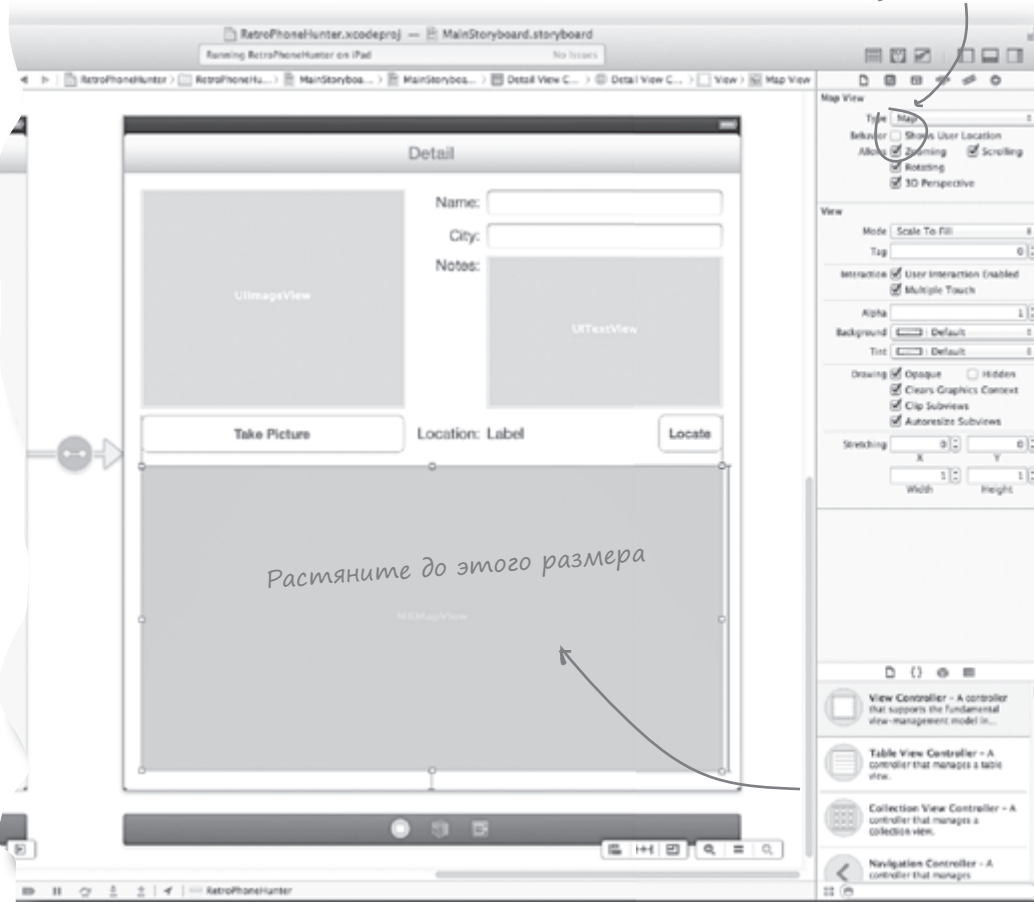
Упражнение
Решение

2

Настройте детализированное представление для отображения карты.

Мы оставили в детализированном представлении место для карты. В файле раскраски перетащите компонент MKMapView в нижнюю половину представления.

Здесь можно разрешить приложению постоянно выводить текущее местоположение (но на это расходуется энергия, поэтому мы так делать не будем).





Упражнение

3

Добавьте ссылки IBOutlet и код для MKMapView.

Теперь, когда вся подготовка выполнена, добавьте ссылки IBOutlet и код Map Kit, обеспечивающий работу карт. Не забудьте создать связь для IBOutlet в Xcode.

```
- (void)configureView
{
    // Обновить пользовательский интерфейс
    // для элемента детализированного представления.

    if (self.detailItem) {
        self.nameField.text = self.detailItem.name;
        self.cityField.text = self.detailItem.city;
        self.notesView.text = self.detailItem.notes;
        self.imageView.image = [UIImage imageWithContentsOfFile:self.
detailItem.imagePath];

        if (self.detailItem.lat != nil && self.detailItem.lon != nil) {
            self.locationLabel.text = [NSString stringWithFormat:@"%%.3f,
%.3f",
                [self.detailItem.lat doubleValue],
                [self.detailItem.lon doubleValue]];
        } else {
            self.locationLabel.text = @"No location.";
        }
    }
}
```



DetailViewController.m



ТЕСТ-ДРАЙВ

Грандиозный финал! Запустите приложение и убедитесь в том, что карта работает!



Эмулятор получает данные позиционирования для сетевого подключения вашего компьютера.



Но на карте нет маркера, показывающего, где находится будка!

Аннотации: придется поработать

Аннотациями называются маленькие флажки, которыми на карте обозначаются точки, представляющие интерес. В чем хитрость? Включение аннотаций потребует реализации протокола аннотаций Map Kit. Map Kit использует протокол аннотаций, чтобы вы могли взять существующие классы и передать их Map Kit напрямую. С другой стороны, это означает, что вам придется добавить *еще немного* кода в класс Phonebooth:

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

#import <MapKit/MapKit.h>

@interface Phonebooth : NSObject <MKAnnotation> {
@private
}
@property (nonatomic, retain) NSString * city;
@property (nonatomic, retain) NSString * name;
@property (nonatomic, retain) NSString * notes;
@property (nonatomic, retain) NSString * imagePath;
@property (nonatomic, retain) NSNumber * lat;
@property (nonatomic, retain) NSNumber * lon;

@property (nonatomic, readonly) CLLocationCoordinate2D
coordinate;

- (NSString *) title;
- (NSString *) subtitle;

@end
```

Протокол MKAnnotation выглядит немного необычно: он определяет свойство и два метода чтения. Они используются MapView для позиционирования маркера и заполнения оверлея, если пользователь прикоснется к маркеру.



Phonebooth.h



Будьте
осторожны!

Еще раз напомним: если вы используете автоматическое генерирование файлов **NSObject**, эти изменения будут потеряны.

Полная реализация протокола аннотаций

Протокол требует наличия свойств coordinate, title и subtitle. Вместо того чтобы синтезировать свойство, вы должны реализовать его самостоятельно, просто возвращая позицию телефонной будки, название и т. д.

В приложении, в котором предполагается более масштабная миграция данных, следует реализовать отдельный класс, реализующий этот протокол, который содержит ссылку на объект Phonebooth (композиция) вместо прямого включения кода в класс Phonebooth.

чтобы ваш класс реализовал протокол аннотаций, реализуйте методы свойств. В них просто возвращайте уже имеющиеся у вас данные по телефонной будке.

```
- (CLLocationCoordinate2D) coordinate {
    return CLLocationCoordinate2DMake(
        [self.lat doubleValue],
        [self.lon doubleValue]);
}

- (NSString *) title {
    return self.name;
}

- (NSString *) subtitle {
    return self.notes;
}

@end
```

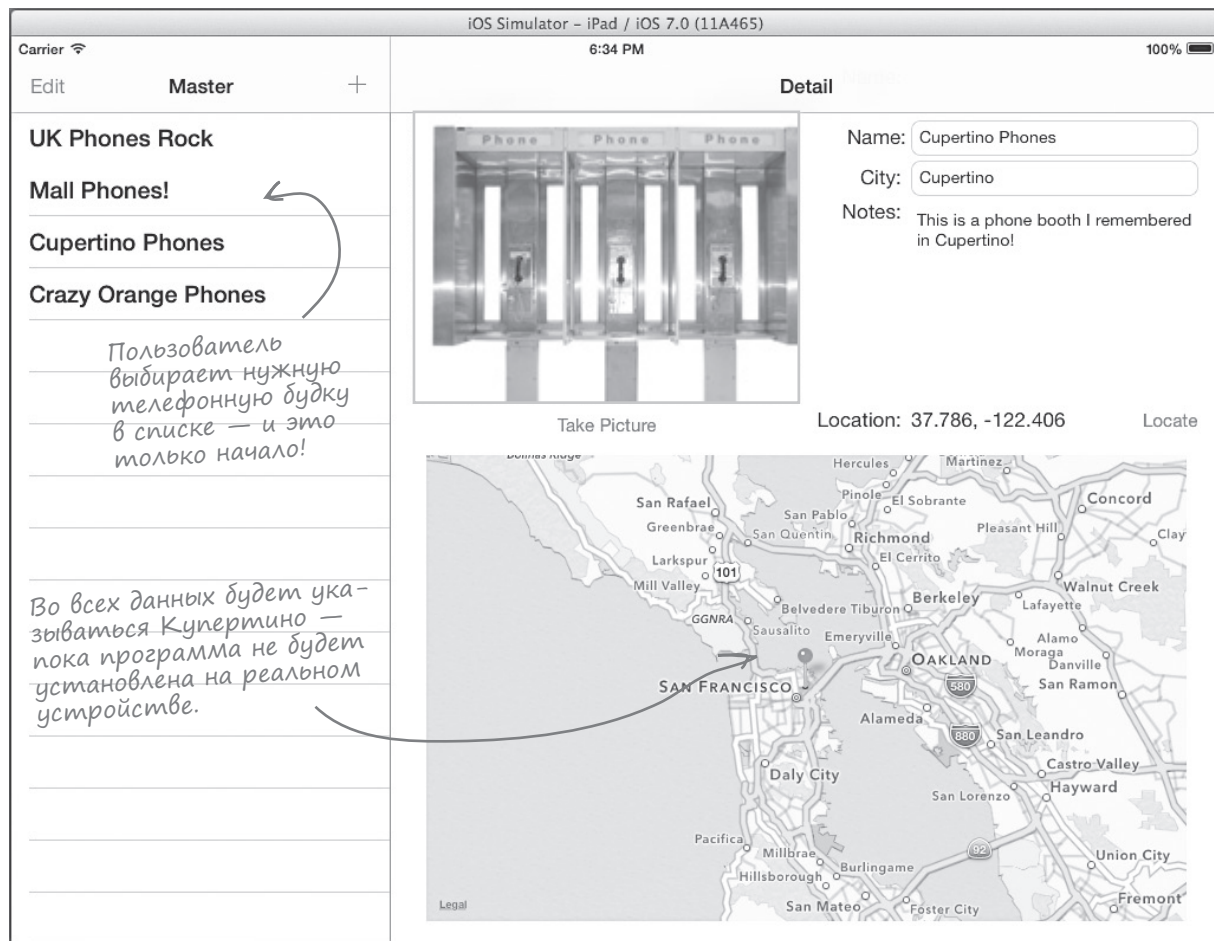


Phonebooth.m



ТЕСТ-ДРАЙВ

Готово! Теперь все должно работать. Начинайте добавлять телефонные будки, и предложите своим друзьям делать то же самое!



Благодаря помощи Core Data вы можете выйти из приложения и перезапустить его — все данные останутся на месте, готовые к использованию!

Go Retro Unlimited

Великолепно!

Вы написали замечательное приложение! Люди загружают его и отправляют нам столько данных о телефонных будках, что мы не верим своим глазам! Прекрасная работа... особенно карты. Кажется, пользователям нравится отыскивать самые экзотические места, в которых установлены телефонные будки.

В знак благодарности высылаем вам нашу фирменную футболку с телефонной будкой, прямо из-под пресса. Носите на здоровье!

Джимми Уэйн,

СЕО





Ваш инструментарий

Глава 8 осталась позади, а ваш инструментарий пополнился навыками работы с устройствами.

UIImagePickerController

- ☐ Управляет камерой.
- ☐ Работает с фотогалереей на устройстве.
- ☐ Содержит встроенные представления.
- ☐ Также может работать с видео.

iPad

- ☐ Требуется поддержка разной ориентации.
- ☐ Иначе использует экранное пространство.
- ☐ Универсальные приложения поставляются с двумя файлами раскэдровки: для планшета и телефона.

Core Location

- ☐ Использует WiFi, GPS и триангуляцию по вышкам сотовой связи для определения текущей позиции устройства.
- ☐ Интенсивно расходует ресурсы (батарею).
- ☐ Спрашивает разрешения пользователя, прежде чем определять его местоположение.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Приложения iOS в основном обращаются к устройству только для чтения.
- Приложения iOS изолируются в структуре файловой системы на устройстве.
- Приложениям разрешены операции чтения и записи с каталогами tmp и cache в «песочнице».
- Каталоги cache и tmp не включаются системой в процесс резервного копирования.

Rails 4. Гибкая разработка веб-приложений

С. Руби, Д. Томас, Д. Хэнссон



ISBN 978-5-496-00898-3

Объем: 448 с.

Перед вами новое издание бестселлера «Agile web development with Rails», написанного Сэмом Руби — руководителем Apache Software Foundation и разработчиком формата Atom, Дэйвом Томасом — автором книги «Programming Ruby» и Дэвидом Хэнссоном — создателем технологии Rails. Rails представляет собой среду, облегчающую разработку, развертывание и обслуживание веб-приложений. За время, прошедшее с момента ее первого релиза, Rails прошла путь от малоизвестной технологии до феномена мирового масштаба и стала именно той средой, которую выбирают, чтобы создавать так называемые «приложения Web 2.0». Эта книга, уже давно ставшая настольной по изучению Ruby on Rails, предназначена для всех программистов, собирающихся создавать и развертывать современные веб-приложения. Из первой части книги вы получите начальное представление о языке Ruby и общие сведения о самой среде Rails. Далее на примере создания интернет-магазина вы изучите концепции, положенные в основу Rails. В третьей части рассматривается вся экосистема Rails: ее функции, возможности и дополнительные модули. Обновленное издание книги описывает работу с Rails поколения 4 и Ruby 1.9 и 2.0.

Идеальная IT-компания. Как из гиков собрать команду программистов

Б. Фитцпатрик, Б. Коллинз-Сассмэн



ISBN 978-5-496-00949-2

Объем: 208 с.

В современном мире разработки ПО успех программиста во многом зависит не только от качества кода, но и от его взаимодействия с другими людьми. В этой занимательной и ироничной книге раскрываются основные закономерности и шаблоны поведения, возникающие в команде разработчиков ПО. Рассматриваются основные роли каждого из участников коллектива, паттерны их поведения и примеры организации наиболее эффективного взаимодействия внутри команды программистов. Эта книга поможет вам оценить важность человеческого фактора в процессе разработки ПО и научиться выстраивать эффективно работающую команду для IT-проекта любой сложности..