Иван Трещев
Программирование для мобильных платформ. IOS

Иван Трещев

ПРОГРАММИРОВАНИЕ ДЛЯ МОБИЛЬНЫХ ПЛАТФОРМ

IOS

Шрифты предоставлены компанией «ПараТайп»

Участие в тестировании разработанных приложений Владислав Андреевич Чусов
Участие в работе над иллюстрациями Мария Сергеевна Аксютина
Участие в верстке и корректуре Анастасия Сергеевна Ватолина

Данная книга обобщает опыт работы лаборатории мобильных приложений на базе ФГБОУ ВО КнАГУ, где автор был ее руководителем. Приложения, разработанные в книге, были успешно выложены в магазин приложений. В книге вы найдете описание основных моментов для разработки приложений.

12+
ISBN 978-5-4493-9973-1
Создано в интеллектуальной издательской системе Ridero

Оглавление

Введение

Разработка мобильных приложений может приносить и стабильный доход и стать точкой роста для профессионала. Современный рынок мобильных устройств полон различными аппаратами всевозможных форм-факторов, если говорить про IOS то это различные телефоны и планшеты фирмы Apple, умные часы и плееры. Конечно по сравнению с другими платформами (скажем Windows Phone или Android) читатель, для того чтобы начать разрабатывать приложения для проприетарной операционной системы для мобильных устройств Apple, должен по крайней мере приобрести ПЭВМ Mac производства фирмы Apple, поскольку Xcode — среда программирования доступна только для MacOS (что является не дешевым удовольствием) и желательно иметь под рукой для тестирования минимальный набор аппаратных устройств — телефон и планшет, хотя в среде программирования и предусмотрены различные эмуляторы, но опыт показал, что все же желательно тестирование проводить на реальных устройствах. Итого инвестиции в процесс разработки приложений порядка 1500€. Помимо всего вышеперечисленного ежегодно необходимо оплачивать 100 долларов США как сбор для разработчиков (к примеру для платформы Android взнос единоразовый), что осуществить так же не просто, особенно для жителей России, поскольку номер факса для связи с Apple именно из США.

Лаборатория которой руководил автор на протяжении 5 лет занималась разработкой различных приложений для самых популярных за последнее пятилетие операционных систем носимых устройств — Android, IOS, Windows Phone. Сегодня платформа IOS насчитывает многомиллионную аудиторию и располагает одним из самых удобных и эргономичных способов для авторов (будь то песни, книги или приложения) для монетизации своих творений при этом не неся затрат на тиражирование, продажу, экспозицию и другие накладные расходы.

По сравнению с другими платформами IOS выделяют несколько немаловажных аспектов. Во-первых эмуляторы есть для всех устройств (не просто для популярных как Windows Phone и просто эмуляторы Android). Во-вторых это конечно то, что ваше приложение действительно будут тестировать специалисты Apple. Причем тестировать будут весьма притязательно. Не стоит надеяться что только разработав приложение и отправив его на модерацию оно тут же будет размещено в магазине (как в случае с Android). Для этого потребуется от двух рабочих дней. В-третьих все же есть единый стиль, единые стандарты программирования, единообразные интерфейсы и механизмы их создания. Member Center, ICloud, синхронизация со всей инфраструктурой Apple. Этот список довольно велик. Причем благодаря постоянным обновлениям операционной системы и «экосистемы» ежегодно появляются новые технологии компании, которые интегрируются в приложения.

В работе не было попытки дать детальное объяснение всех возможных аспектов программирования под IOS. А скорее данная книга будет полезна тем кто все же открыл Xcode приобрел аккаунт разработчика и планирует начать пробираться через тернии и хитросплетения современных технологий программирования.

Данная книга посвящена разработке приложений именно под платформу от Apple и является второй в цикле, над которыми автор работает в настоящее время.

У читателя предполагается опыт программирования на объектно-ориентированном языке, желательно опыт на C#.

Автор хотел бы выразить огромную благодарность Чусову В. А. и Аксютиной М. С. без которых невозможно было работать.

Основы работы с Xcode

Сегодня мы разберемся, как создать свой проект, познакомимся с основными объектами Фреймворка UIKit.

И так, первым шагом мы запустим Xcode, и создадим проект, после этих действий, должно появиться окно, где можно выбрать уже подготовленный для чего-либо проект,

например, выберем «Single View Application» (рис. 1.1). С остальными типами познакомимся позже.

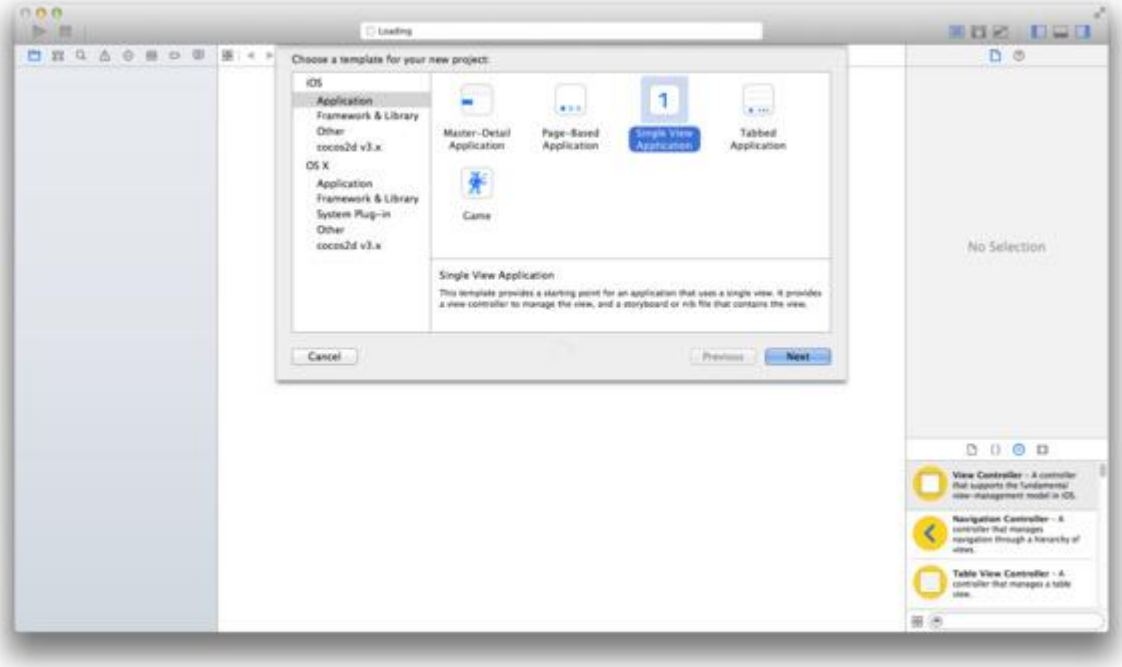


Рисунок 1.1 — Выбор типа приложения

Далее вводим данные, название проекта, название компании, уникальный идентификатор компании, язык на котором мы будем программировать, выбираем Objective-C, далее выбираем для какого типа устройств создаем приложением, и последний пункт — Core data, это некая оболочка/Фреймворк для обработки данных. Пока что, вы можете вводить любые данные, т.к. этот всего лишь тестовый проект.

После проделанных манипуляций, перед нами должно появится окно с настройками нашего проекта (рис. 1.2).

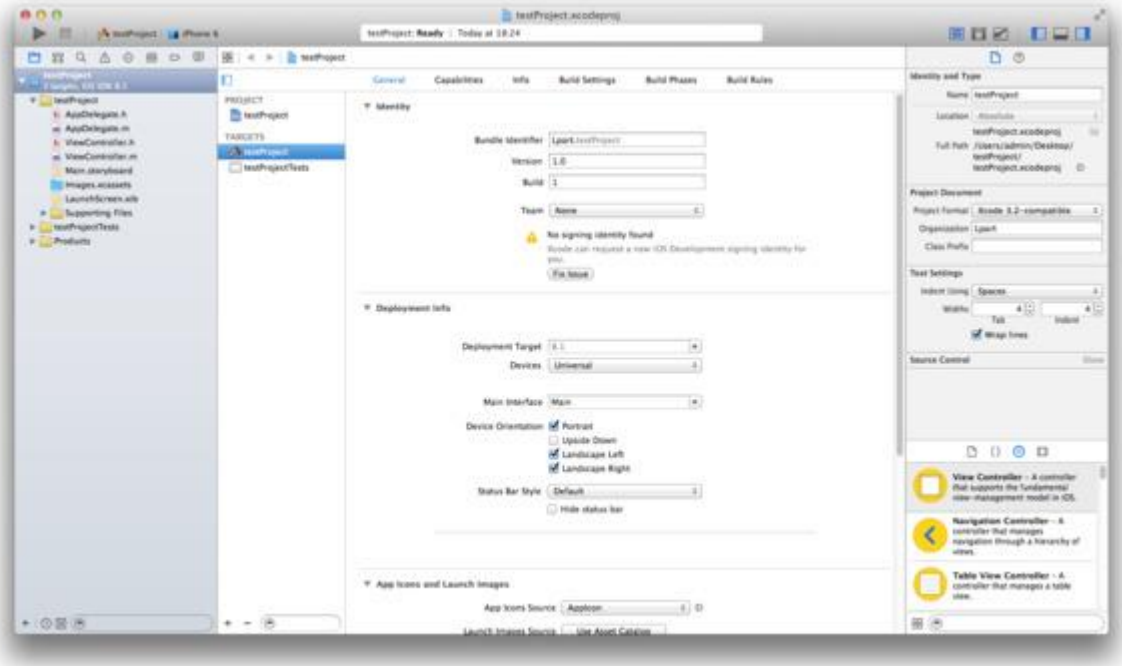


Рисунок 1.2 — Настройки приложения

Разберемся с targets. Грубо говоря, один target — один вид нашего проекта. В каждом target мы можем добавлять или убирать какие-либо объекты, данные. К примеру, мы создаем

приложение для двух компаний, но с разными картинками, мы в одном target вставляем одни картинки, а в другом иные картинки. Так же есть еще такое понятие Scheme — это схема, которую мы можем изменять, создавать свою новую или создать автоматическую Она позволяет нам настраивает запуск приложения, его сборку и т. д.

Далее идут основные параметры проекта (рис. 1.3). Identity. В нем предоставляется информация о проекте и команде разработчиков. Deployment info — здесь мы указываем, какие версии iOS будет поддерживать наше приложение, для какого устройства разработка ведется, какие допустимы ориентации устройства. Status Bar — это строка состояние устройства.

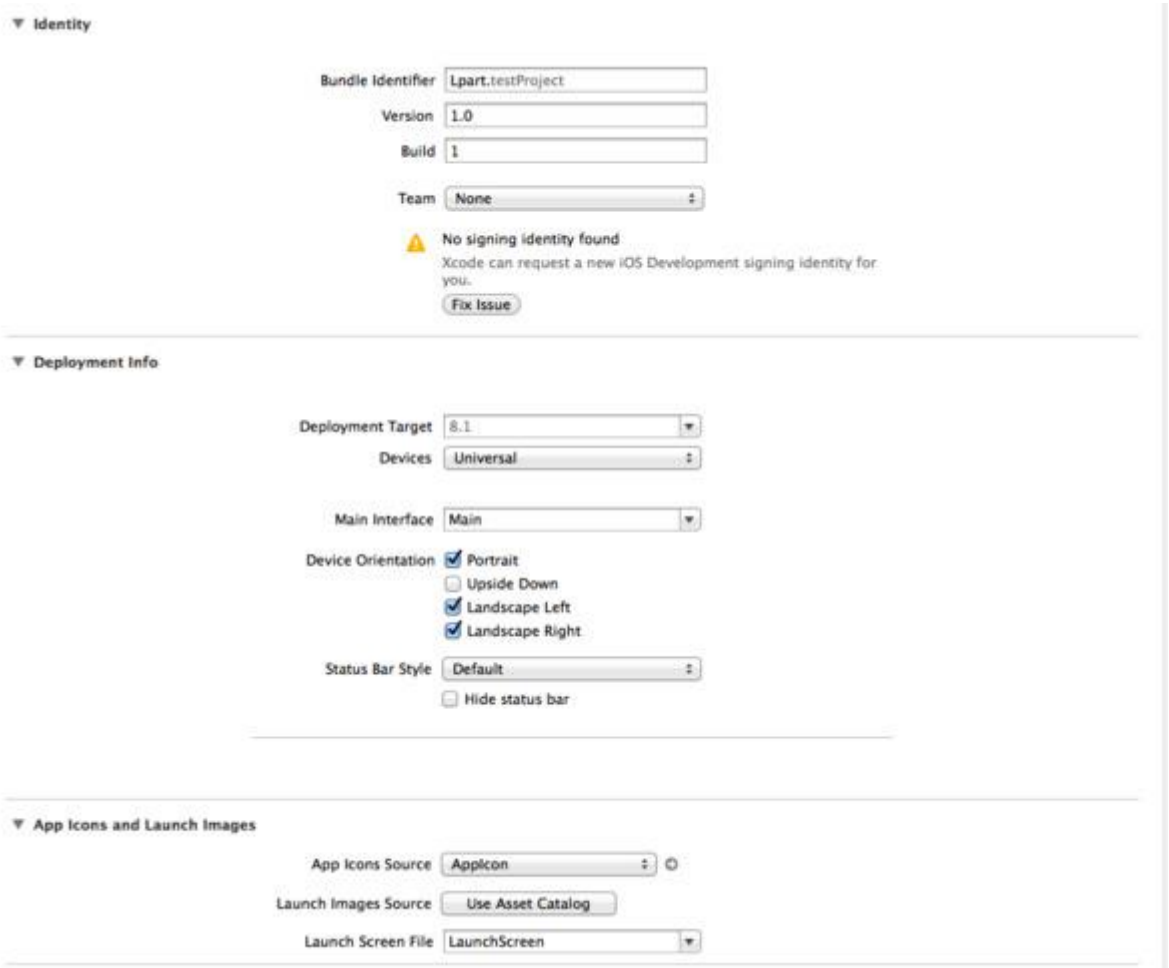
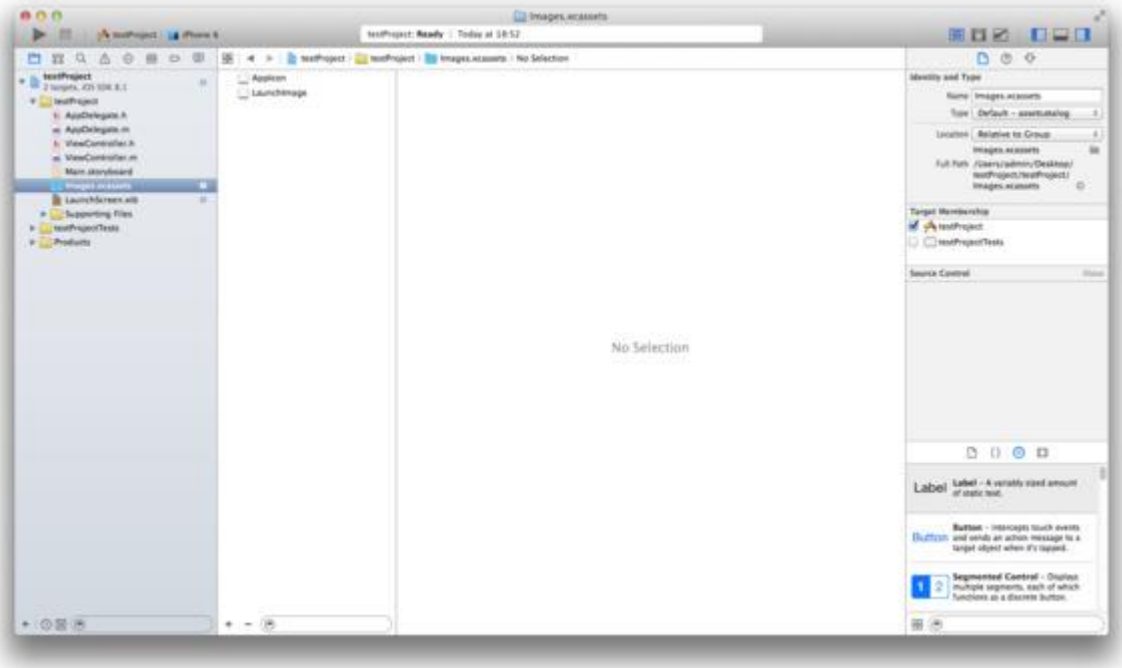


Рисунок 1.3 — Основные параметры проекта

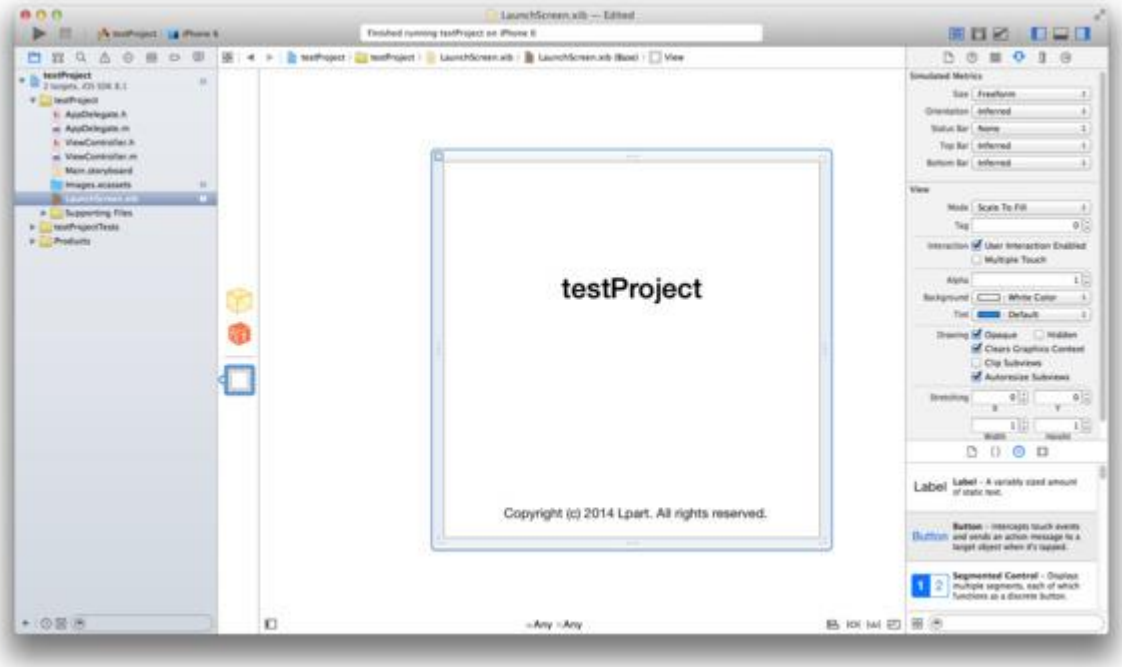
Затем указывается галерея (рис. 1.4) иконок и загрузочных экранов приложения.



Р

исунок 1.4 — Images. xcassets

Загрузочный экран указывается не изображение, а xib-файл (рис. 1.5), в котором у нас находится один объект UIKit — View, попозже разберемся что это такое. Какая разница между xib и storyboard, второе используется в версиях iOS 5 и позднее, xib является устаревшим механизмом разработки интерфейса программы, еще одно отличие — это то, что значительно упростилась работа по созданию интерфейса, различие в представление иерархии объектов, также значительно сократилось кол-во строчек xml-кода.



Р

исунок 1.5 — Launch screen

Вернемся к настройкам проекта, рассмотрим еще параметр info (рис. 1.6). В нем содержаться основные данные о проекте, которые хранятся в файле "Info.plist», в нем мы можем указывать параметры нашего проекта. Вкладка Capabilities, содержит возможности приложения, такие как: Game Center, iCloud, Maps и т. д. В этой кладке мы подключаем дополнительные возможности. Далее Build Settings — настройка сборки приложения. Build

Phases — позволяет нам настраивать сборку приложения для текущего target. Build Rules — позволяют нам задавать правила сборки приложения, к примеру, в момент сборки, сжимать текстовые файлы с помощью определенных скриптов.

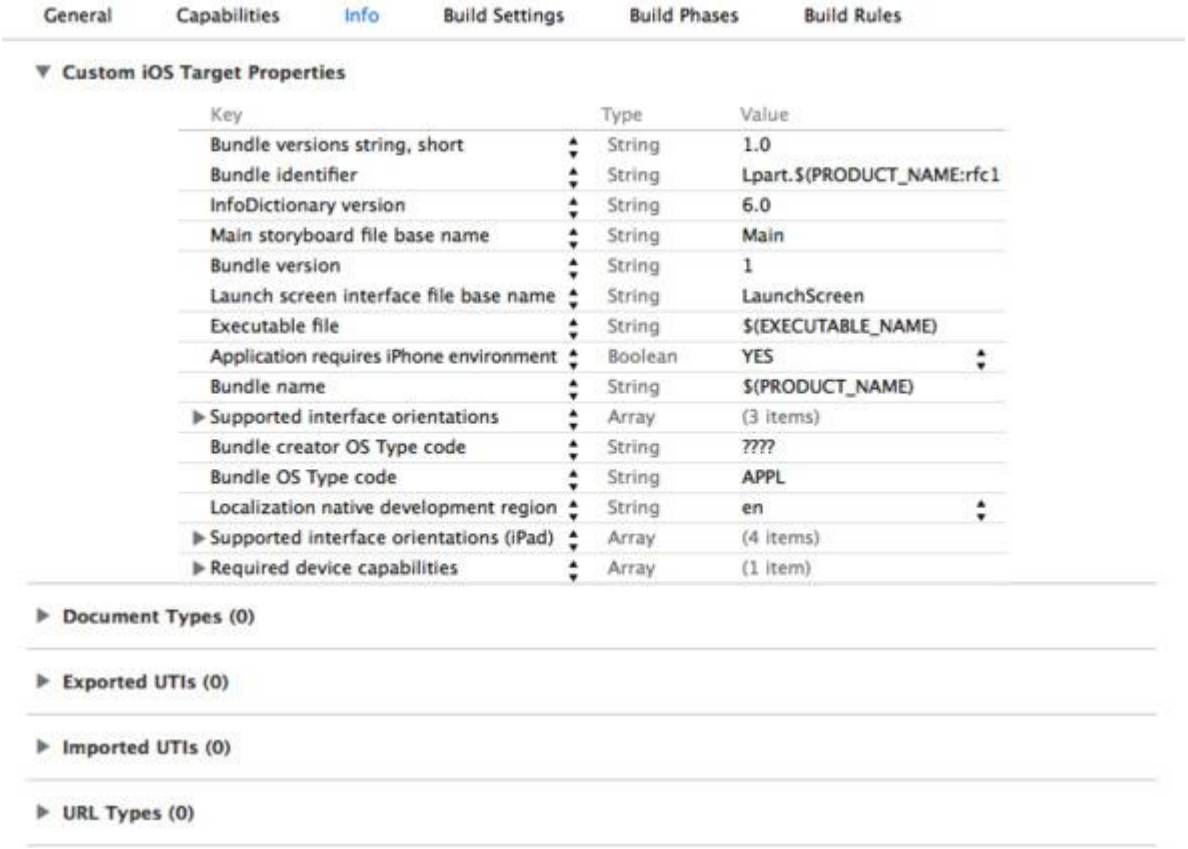


Рисунок 1.6 — Информация о проекте

Теперь перейдем в Storyboard. Как уже упоминалось ранее, это есть механизм разработки интерфейса приложения, также мы можем создавать интерфейса приложения в коде.

Перед нами должен появится View Controller (рис. 1.7) — это фундаментальный объект в UIKit, на котором можно отображать различные объекты UIKit. Сам View Controller добавлен на объект UIWindow, который поддерживает отображение графических элементов на экран. Как вы можете заметить у вью есть три элемента. Первый элемент — означает, что выбран сам вью. Второй элемент — открывает нам список готовых действий. Третий элемент — высвобождает из стека предыдущий вью, тем самым возвращает предыдущий «экран».

Рисунок 1.7 — Storyboard

Рассмотрим элементы Storyboard, в нижней части экрана (рис. 1.8) указываются размеры экранов, с которыми можно работать. Начиная с 3.5-дюймового телефона в портретной ориентации, и заканчивая 12.9-дюймовым планшетом в любой ориентации.

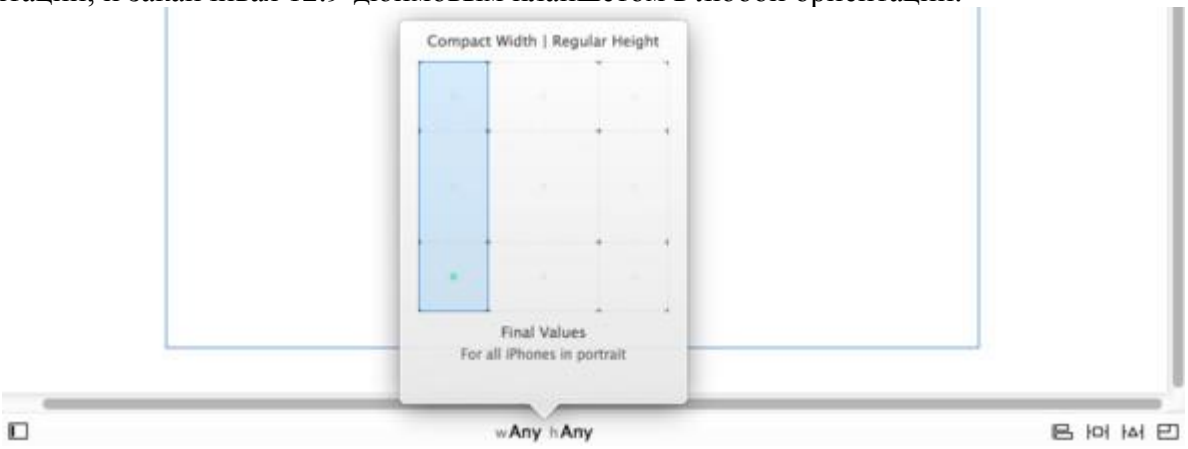


Рисунок 1.8 — Настройка представления

В левой нижней части есть кнопка, которая открывает нам иерархию объектов на View Controller, в правой нижней части, начиная от середины, первая кнопка позволяет выравнивать объекты на представление. Вторая кнопка позволяет нам закреплять объекты на определенной позиции экрана. То, что мы добавляем выравнивание или закрепляем объект — называется добавлением constraint. Третья кнопка обновляет constraint, либо изменяет положение объектов в соответствие с их constraints. Последняя кнопка обновляет constraints, если изменяются размеры экрана.

Теперь рассмотрим правый блок Xcode (рис. 1.9).

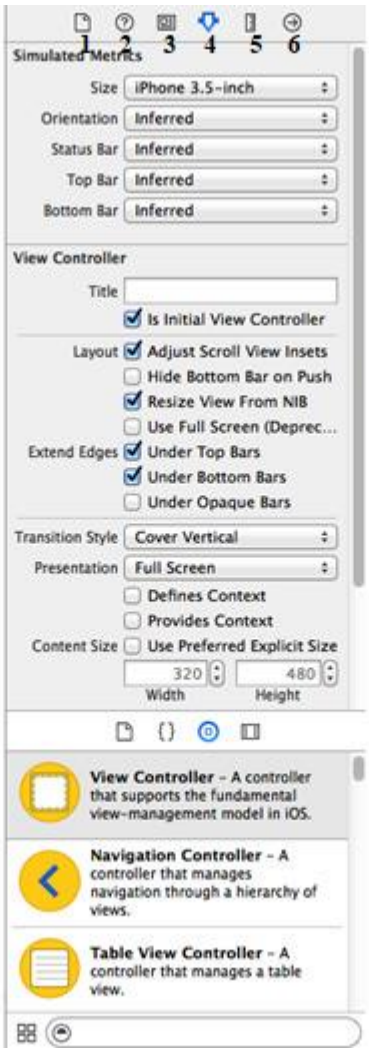


Рисунок 1.9 — Утилиты

Этот блок содержит сведения и параметры, выделенного объекта, снизу, находится библиотека объектов, список различных конструкций кода, файлов, добавляемых в проект, и медиа библиотека.

Разберем пронумерованные вкладки (рис 1.9). Первая — файловый инспектор, в котором указаны характеристики storyboard, такие как: где расположен файл, под какую версию разрабатывается и т. д. Вторая — быстрый помощник. Третья — идентификатор выбранного объекта, в нем задается идентификатор объекту, указывается класс. Четвертая вкладка — параметры объекта. В пятой вкладке, настраиваются размеры объекта. Шестая — показывается существующие связи между объектами.

Рассмотри навигацию проекта (рис. 1.10). Первая вкладка — файловый инспектор проекта. Вторая — показывает нам классы, а при их раскрытие, показывает методы классов. Третья — поисковая строка, ищет во всем проекте, также можно искать и в самом классе (cmd+f). Четвертая — показывает список предупреждений и ошибок, если таковые имеются. Пятая — список существующих тестов для приложения. Шестая вкладка — показывает нагрузку на процессор, память, сеть, файловую систему телефона. Седьмая — список точек остановок. Восьмая — показывается выполненные действия над проектом.

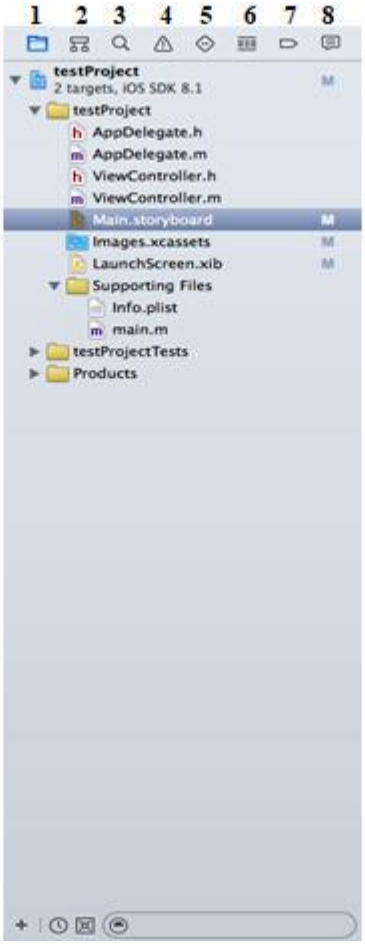


Рисунок 1.10 — Навигация проекта

Теперь добавим кнопку в наш проект. Для этого, нам необходимо найти библиотеку объектов, и в ней найти Button (рис 1.11).

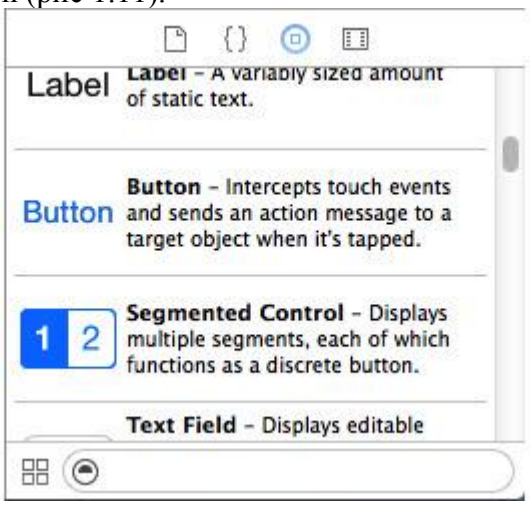


Рисунок 1.11 — Button

Затем, с помощью метода «drag and drop», добавляем на наш View Controller. Затем таким же способом добавим Label. Обоям объектам установить размер 200x40, в инспекторе размера. А нашему вью в инспекторе параметров, установите для size значение — iPhone 3.5 inch.

Теперь мы должны установить связи, между кнопкой и вью, и между надписью и вью. Нам нужно разделить экран Xcode на две части, нажмем на соответствующую кнопку в правом верхнем углу (рис. 1.12).

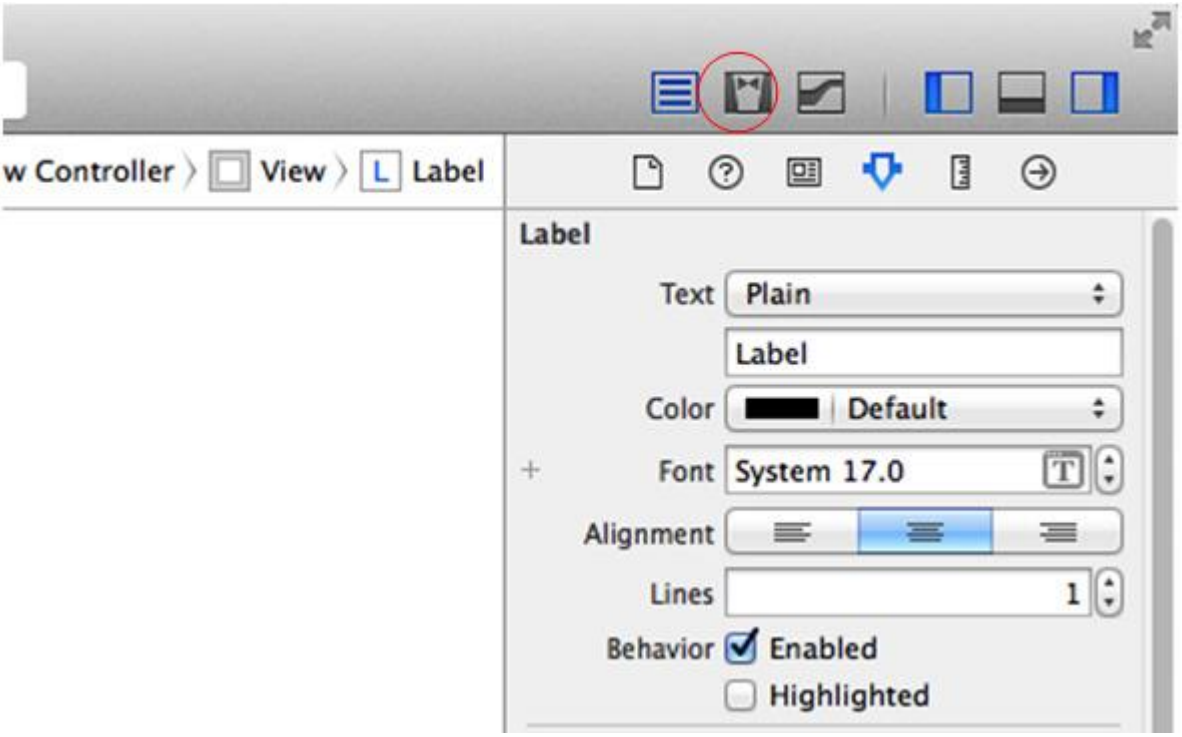


Рисунок 1.12 — Assistant Editor

Теперь у нас есть два окна, в левой части storyboard, в правой части какой-либо класс. Теперь в правую часть поместим ViewController. h, из файлового инспектора перетянем этот класс в правое окно (рис. 1.13).

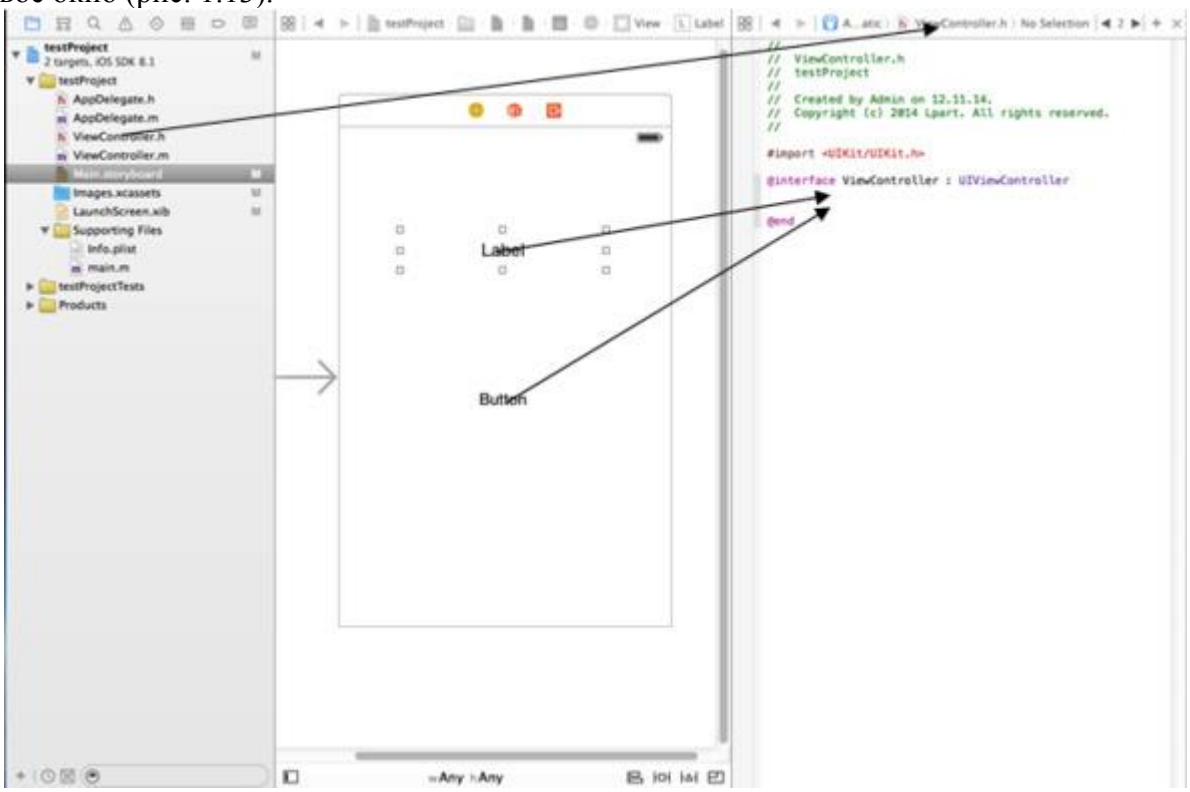
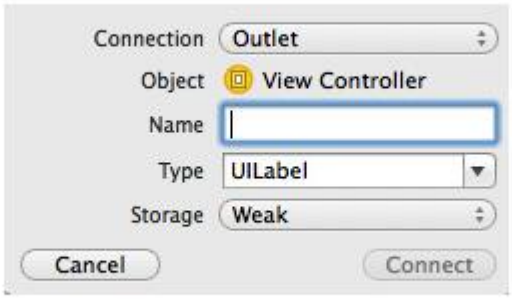


Рисунок 1.13 — Assistant Editor II

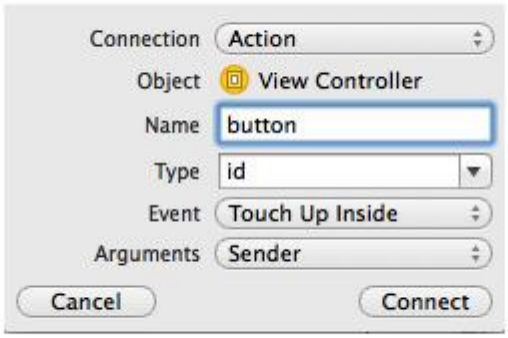
И добавим связи в представление вью. Правой кнопкой мыши перетяните стрелку с Label и Button в ViewController. h. Перед вами появится окно с параметрами (рис. 1.14), с ними мы разберемся позже, а пока что введите имя надписи.



Р

исунок 1.14 — Параметры свойства

Для Button нужно будет дважды проделать такое действие, первое мы добавляем свойство для кнопки, а второе мы добавляем для него метод, чтобы добавить метод, нужно указать в параметре connection — action (рис 1.15).



Р

исунок 1.15 — Action button

Должно получится следующим образом (рис. 1.16).

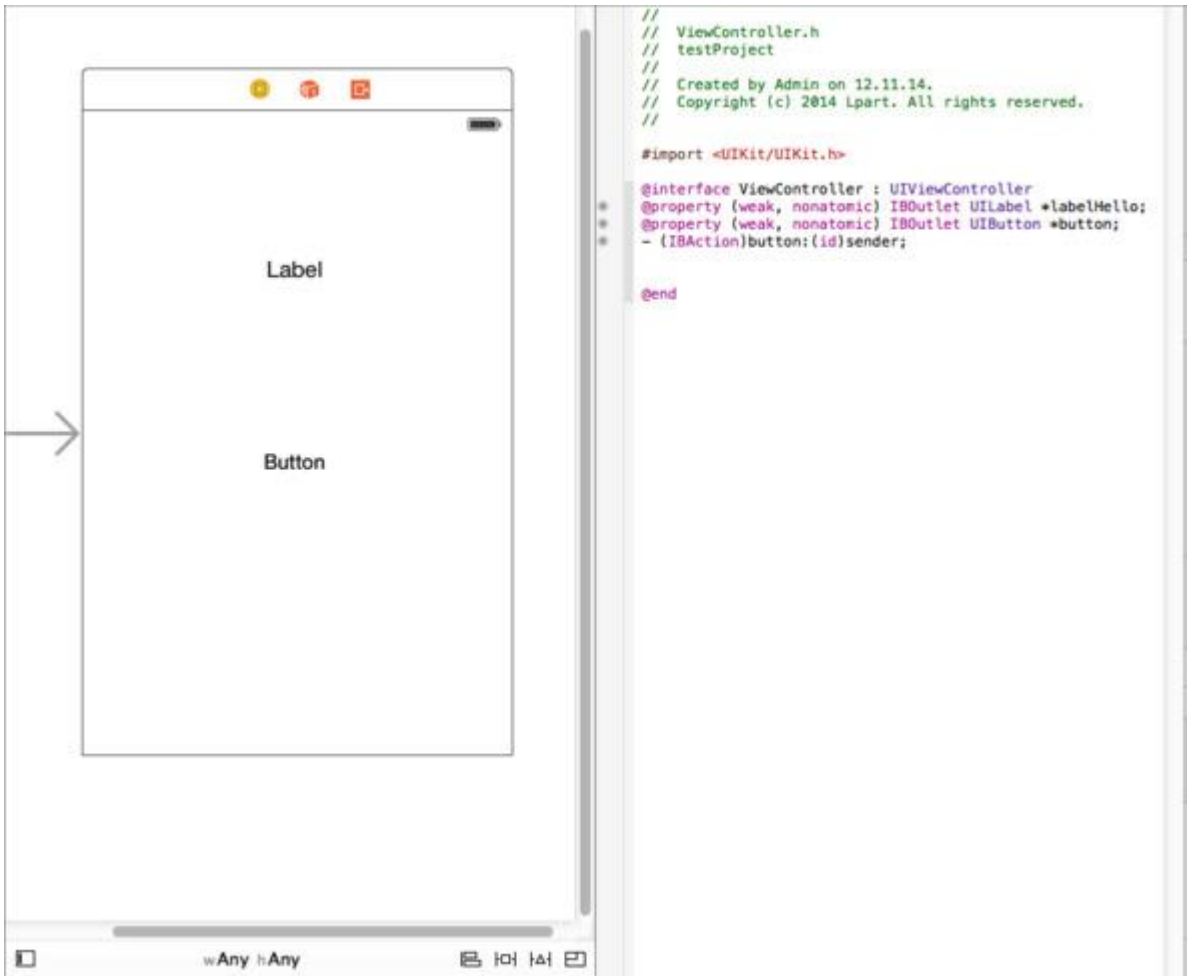


Рисунок 1.16 — Связь объектов и вью

Теперь закроем второе окно и перейдем в класс ViewController. m. Найдем в нем метод — (IBAction) button: (id) sender. Это метод кнопки, который вызывается каждый раз, когда мы нажимаем на кнопку. Пропишем в этом методе следующий код:

— (IBAction) button: (id) sender {
_labelHello. text = @«Hello World!»;
}

Теперь, когда будете нажимать на кнопку, надпись будет менять свой текст на «Hello World», результат работы — рисунок 1.17.

Рисунок 1.17 — Результат работы

И последнее, что осталось — отладчик. Чтобы поставить точку остановки, необходимо нажать в выделенной полосе слева от кода (рис. 1.18).

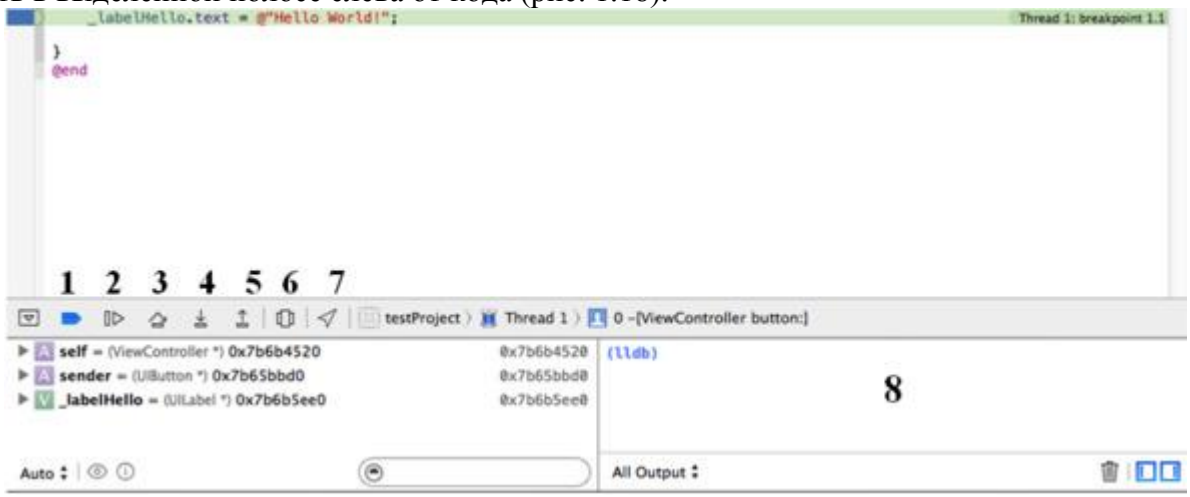


Рисунок 1.18 — Отладчик

Первая кнопка — отключает/включает точки остановки. Вторая — продолжить/приостановить процесс работы приложения. Третья — пошаговая отладка кода. Четвертая кнопка — заход в метод. Пятая — выход из метода. Шестая — позволяет нам просматривать наше приложение в виде 3D-модели. Седьмая кнопка — задает местонахождение. Окно под номером 8 — консоль, в которой выводятся системные сообщения.

Основы графического дизайна интерфейсов приложений

«Быть дизайнером — значит не просто собирать разрозненные элементы воедино, упорядочивать их или как-то изменять. Тут нужно и создавать некую ценность, и придавать смысл, и освещать, и упрощать, и трансформировать, и облагораживать, и сгущать краски, и убеждать, и даже в какой-то мере развлекать».

— Пол Рэнд (Paul Rand)

Перед тем как начать разрабатывать интерфейс приложения, следует разобраться с его основными задачами. Грамотно созданная графическая составляющая приложения должна способствовать эффективной работе, облегчать взаимодействие пользователя с ПО. Исходя из этого, сформулируем 20 основных тезисов, которые должен знать каждый дизайнер:

1.Обязанность интерфейса — обеспечение взаимодействия.

Интерфейсы служат для обеспечения взаимодействия между людьми и окружающим миром. Они помогают нам прояснять, освещать, реализовывать и наблюдать взаимосвязи; они могут объединять и разъединять нас, влиять на наши ожидания; а кроме того, они дают нам доступ к различным услугам. Интерфейсы призваны выполнять определенные функции.

2. Ясность прежде всего.

Любой интерфейс в первую очередь должен быть понятным. Чтобы эффективно использовать разработанный вами интерфейс, люди должны понимать, что он из себя представляет, зачем им его использовать, какие задачи они смогут с его помощью выполнять, к чему приведет то или иное действие — и тогда они смогут успешно с ним взаимодействовать. Да, разобраться в интерфейсе с первого раза может быть непросто, но двусмысленностям в нем места нет. Понятному интерфейсу пользователи доверяют и поэтому, скорее всего, будут использовать его в дальнейшем. В общем, вместо того чтобы запихивать все на один экран и тем самым запутать пользователей, лучше сделайте сотню понятных экранов.

3. Внимание любой ценой.

Внимание пользователей бесценно. Дайте ему сосредоточиться, не засоряйте приложения элементами, отвлекающими внимание от главного назначения того или иного созданного вами экрана. Если пользователям нужно что-то прочитать, то дайте им достаточно времени для этого, а уж потом показывайте рекламу, если это необходимо. Цените внимание ваших пользователей: от этого в выигрыше будут и они, и вы. Для обеспечения использования продукта внимание пользователей — жизненно важный фактор. Старайтесь удерживать его любой ценой.

4. Под контролем пользователей.

Людям нравится чувствовать контроль над ситуацией. Многие разработчики об этом не задумываются, и в результате пользователи вопреки своему желанию вынуждены совершать операции, которые они не собирались совершать, причем направление их движения оказывается весьма неясным, а результаты действий — неожиданными. Дайте пользователям почувствовать, что ситуация под их контролем, периодически отображая состояние системы, описывая причинно-следственные связи (если вы сделаете это, случится то-то) и помогая им ясно понять, чего можно ожидать от каждой конкретной операции.

5. Лучшее управление — прямое управление.

Лучший интерфейс — никакого интерфейса: например, с объектами реального мира мы взаимодействуем напрямую. Но постепенно появляется все больше объектов цифровой природы, которыми управлять напрямую невозможно, и тут нам на помощь приходят интерфейсы. Очень легко сбиться с верного пути и в итоге перегрузить интерфейс кнопками, финтифлюшками, графикой, параметрами, настройками, окнами, дополнительными вставками и прочим «мусором». В результате пользователи вынуждены вместо выполнения задач заниматься управлением элементами интерфейса. Чтобы этого избежать, возьмите за образец прямое управление: интерфейс должен быть максимально незаметным и способным распознавать естественные человеческие жесты. В идеале у пользователей должно появиться ощущение, будто они управляют объектом напрямую.

6. Один экран — одна основная задача.

Каждый экран приложения должен служить для выполнения какой-либо одной задачи, стоящей перед пользователями. Так пользователям приложения будет проще в нем разобраться и с ним работать, а разработчикам — расширять его функционал при необходимости. Экраны, поддерживающие выполнение нескольких основных задач, сбивают пользователей с толку. Очевидно, что, например, текст не может содержать две или три главные темы — главная тема может быть только одна. Так и дизайнерам следует закладывать в каждый экран возможность выполнения только одной ключевой задачи.

7. Второстепенная задача, знай свое место.

Кроме какой-то одной ключевой задачи экраны также могут служить для выполнения нескольких второстепенных задач. Но важно помнить, что главные и второстепенные задачи нельзя валить в одну кучу. Второстепенные задачи не должны выходить на первый план: к примеру, они могут быть оформлены менее заметным образом или отображаться только после того, как была выполнена ключевая задача.

8. Место для шага вперед.

Так как большинство операций пользователей не обрывают процесс взаимодействия, а последовательно переходят друг в друга, весьма благоразумно будет спроектировать для каждой операции какое-нибудь продолжение. Постарайтесь представить себе, каким будет следующий шаг пользователей в каждом конкретном случае, и выстраивайте интерфейс в соответствии с этим. Как и в обычно человеческом общении, здесь нужна отправная точка для дальнейшего взаимодействия. Если пользователи уже сделали все, что требовалось, не бросайте их: дайте им возможность естественным образом сделать следующий шаг на пути к достижению их целей.

9. Поведение определяет внешний вид.

Люди предпочитают иметь дело с тем, что ведет себя предсказуемым образом: это равным образом относится к другим людям, животным, объектам — и в том числе к программному обеспечению. Когда чье-то поведение совпадает с нашими ожиданиями, мы чувствуем себя на правильной волне. Соответственно, элементы дизайна интерфейса должны выглядеть соответственно своему поведению. На практике это означает, что пользователи должны понимать, как поведет себя тот или иной элемент интерфейса, едва взглянув на него. Элемент, похожий на кнопку, и вести себя должен как кнопка. Не стоит заигрывать с основополагающими аспектами взаимодействий пользователей и интерфейса — приберегите свою творческую энергию для задач другого порядка.

10. Как важно быть последовательным.

Из предыдущего пункта следует, что если поведение экранных элементов

различается, то и выглядеть они должны по-разному. Безусловно, элементы, схожие в поведении, и выглядеть должны схожим образом (последовательно). Но не менее важно, чтобы несхожие элементы были оформлены по-разному (т. е. непоследовательно). Дизайнеры-новички, стараясь сделать интерфейс логичным и последовательным, зачастую игнорируют существенные различия между элементами и используют для их оформления одни и те же приемы там, где следовало бы внести разнообразие.

11. Четкая иерархия.

Четкая визуальная иерархия достигается, когда элементы на экране расположены в определенном порядке. То есть одни и те же элементы отображаются в одном и том же порядке каждый раз. Плохо проработанная визуальная иерархия не приносит никакой пользы и только сбивает пользователей с толку. В постоянно изменяющихся средах не так-то просто поддерживать четкую иерархию элементов, потому что визуальный вес становится относительной величиной: ведь если выделено все, то не выделено ничего. Если на экран нужно добавить заметный элемент, дизайнеру может понадобиться сделать все остальные элементы менее заметными, чтобы сохранить визуальную иерархию. Большинство пользователей не задумываются о визуальной иерархии при работе с интерфейсом, но при этом ее продуманное (или непродуманное) выстраивание — это один из самых легких способов улучшить (или ухудшить) дизайн.

12. Грамотная организация снижает когнитивную нагрузку.

Джон Маэда (John Maeda) в своей книге Simplicity пишет, что грамотная организация элементов интерфейса позволяет придать экрану менее загруженный вид. С помощью продуманной организации элементов вы сможете продемонстрировать связи между ними, и освоить такой интерфейс пользователям будет куда проще. Группируйте схожие элементы, располагайте их на экране таким образом, чтобы пользователям стало понятно, как они связаны между собой. Благодаря грамотной организации контента можно значительно снизить когнитивную нагрузку пользователей. Если в самом дизайне будут наглядно продемонстрированы связи между элементами, пользователям уже не придется мучительно в них разбираться. Не заставляйте пользователей лишний раз напрягаться — лучше просто покажите им все эти связи между элементами интерфейса с помощью вашего дизайна.

13. Подсказывай, а не указывай: роль цвета.

Цвет физических объектов меняется в зависимости от освещения. Одно и то же дерево, например, в полдень и на закате выглядит совершенно по-разному. В общем, в мире физических объектов цвет включает в себя множество оттенков и вообще довольно относителен, и в интерфейсах цвет также должен играть соответствующую роль. Цветом можно выделять объекты, привлекая к ним внимание пользователей, но при этом элементы нельзя разделять только по цвету. Если предполагается, что пользователи будут работать с каким-либо элементом продолжительное время, или же элемент содержит объемный текст, рекомендуется использовать для оформления бледные или приглушенные тона — а яркие оттенки приберегите для расстановки акцентов. Разумеется, можно использовать яркие цвета и для фоновой заливки, но только там, где это уместно.

14. Не все сразу.

На каждом экране должно отображаться только самое необходимое. Если пользователям требуется сделать выбор, предоставьте им ровно столько информации, сколько им для этого нужно. Подробностям можно посвятить последующие экраны. Не надо пытаться объяснить все от А до Я или выложить всю информацию разом. По возможности распределяйте рабочий процесс на несколько экранов, раскрывая информацию постепенно. Благодаря этому взаимодействие с интерфейсом остается ясным и понятным для пользователей.

15. Подсказывай с умом.

В идеальных интерфейсах подсказки не нужны вовсе, потому что такой интерфейс легко изучить и использовать. Но если спуститься с небес на землю, то в идеале подсказки должны быть контекстно-зависимыми и появляться только тогда и там, где они нужны, в остальное время оставаясь скрытыми. Заставляя людей открывать справку и искать ответы на возникшие у них вопросы, вы затрудняете их работу с интерфейсом, так как в этом случае им приходится формулировать, что именно они хотят найти. Лучше встраивать подсказки там, где они могут потребоваться. Только убедитесь, что они не будут лишний раз маячить перед носом тех пользователей, которые уже знакомы с интерфейсом.

16.Стартовая страница.

Дизайнеры часто упускают из вида такой важный момент, как первое знакомство с интерфейсом. Чтобы помочь пользователям как можно быстрее освоиться, дизайнер должен работать с прицелом на нулевое состояние — тот момент, когда еще ничего не произошло. Первый экран, который видят пользователи, не должен быть пустым, аки чистый лист, — на нем должны содержаться указания и подсказки для быстрого начала работы. Большинство затруднений при работе с интерфейсом возникает на почве непродуманного нулевого состояния. Но стоит пользователям понять правила игры, как их задача сразу значительно упрощается.

17. Текущие проблемы — главные проблемы.

Пользователям нужно решать задачи, актуальные на данный момент, а не гипотетические вопросы, которые могут возникнуть в будущем. Таким образом, интерфейс, ориентированный на решение потенциальных проблем, никому не будет нужен: изучайте текущую ситуацию и разрабатывайте интерфейс в соответствии с актуальными

проблемами. Витать в облаках и строить гипотезы, конечно, гораздо увлекательнее, но зато результаты вашего труда окажутся востребованы благодарными пользователями, а не отправлены на свалку бесполезных интерфейсов.

18. Лучший дизайн — невидимый дизайн

Интересный факт: действительно хорошие дизайны обычно никак не отмечаются пользователями, работавшими с ними. Причина заключается в том, что удачный дизайн позволяет пользователям сконцентрироваться на их задачах, а не на работе интерфейса. Пользователи, успешно выполнившие свои задачи, не станут задумываться над тем, как это так у них все хорошо получилось. Получается, что пользователи обращают внимание на дизайн только в том случае, если у них возникают какие-либо трудности. Да, дизайнеры не в восторге от того, что за удачные решения их никто не хвалит, но действительно хорошим специалистам вполне достаточно того, что их дизайном активно пользуются. Они понимают, что довольный пользователь — это молчаливый пользователь.

19. Расширяем кругозор

Визуальный и графический дизайн, полиграфия, копирайтинг, информационная архитектура и визуализация — все это входит в дизайн интерфейсов. С этими дисциплинами можно знакомиться вскользь, а можно углубиться в их изучение. Черпайте в них полезные знания — и вперед. Не брезгуйте и на первый взгляд абсолютно не связанными с дизайном интерфейсов сферами.

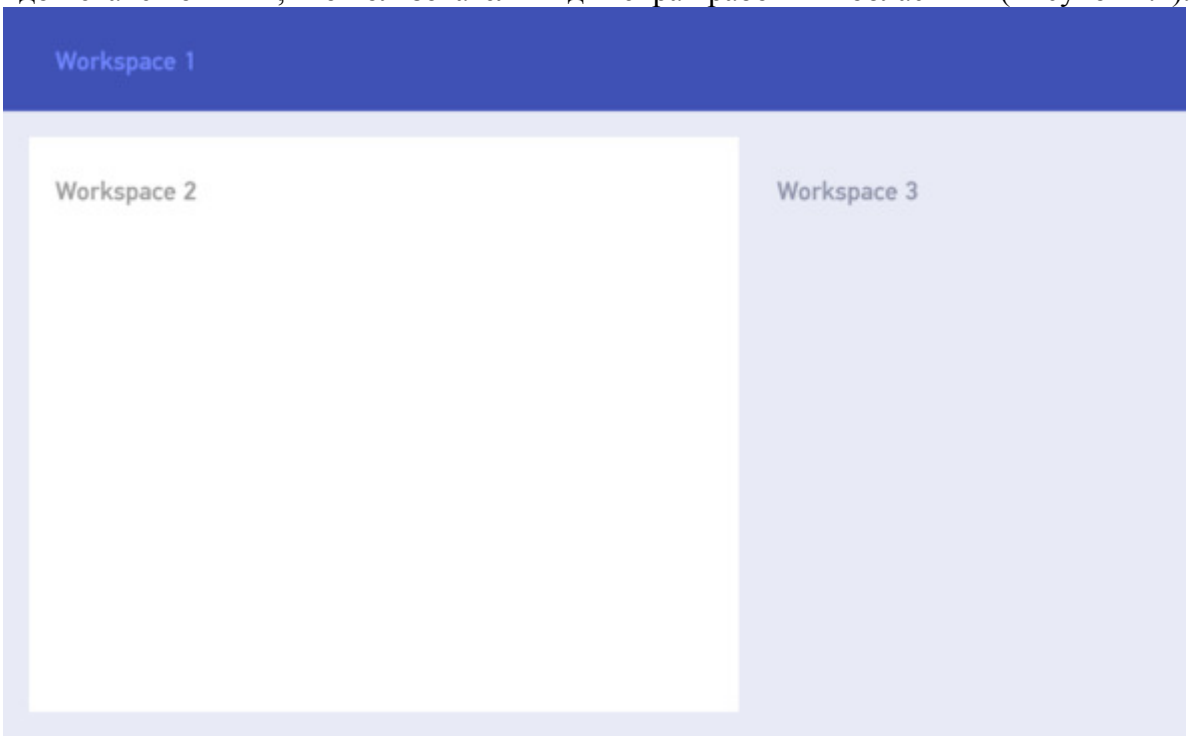
20. Неиспользуемый интерфейс — плохой интерфейс

Как и в других областях дизайна, в дизайне интерфейсов успешным считается тот результат дизайнерских трудов, который оказывается востребован пользователями. Люди не сядут даже в самое красивое кресло, если оно окажется неудобным, и этот предмет мебели не выполнит свою функцию, как и дизайн, который пользователи обходят вниманием. Таким образом, в дизайне интерфейсов важную роль играет создание не только самого объекта, но и некоей среды его использования. Дизайнер создает интерфейс не для услады собственных очей, а для того чтобы им пользовались.

Цвет в дизайне

Цвет — это метод создания баланса элементов.

Во время продумывания сценария взаимодействия и восприятия интерфейса следует на каждом этапе помнить, что пользователь видит экран рабочими областями (Рисунок 2.1).



Р

исунок 2.1. Рабочие области простого интерфейса

Характерно, что последовательность восприятия, у большинства пользователей, будет несколько отлична от нумерации рабочих областей, а именно: workspace 2 → workspace 1 → workspace 3, при этом workspace 2 и 3, очевидно, будут восприниматься как основная и второстепенная часть одного сценария.

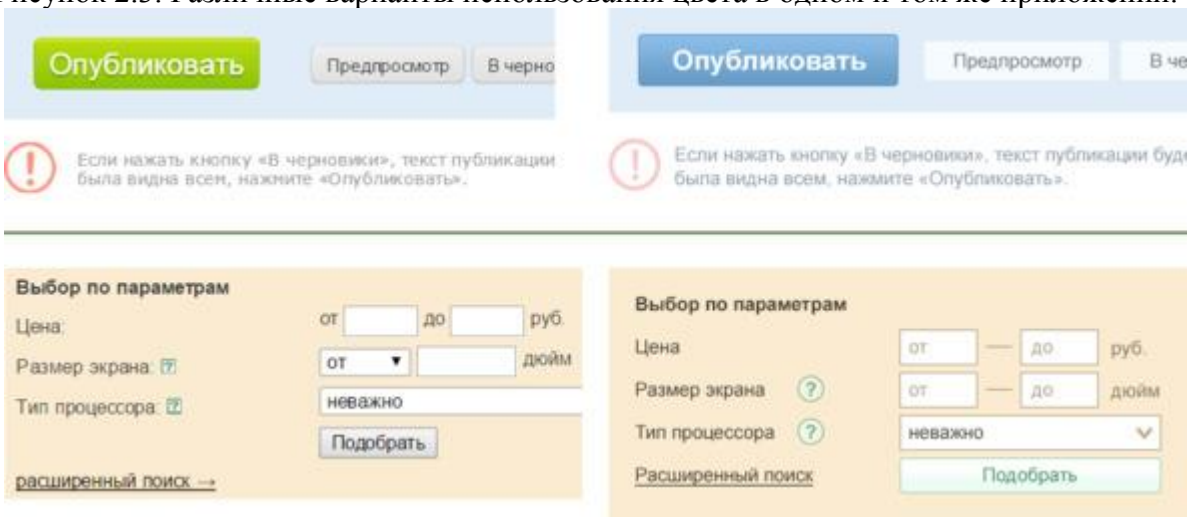
Пользователь стремится игнорировать элементы, находящиеся в другой рабочей области, во время работы с текущей — подсознательно он представляет, что где-то там находится ряд функций, которые ему могут понадобится, но сознательно он их «не видит». Отсюда можно сделать вывод, что элементы внутри областей должны находиться на одном уровне интенсивности относительно других областей. Например на рисунке 2.2 очевидно, что в первом варианте сбивается изначально заданный баланс: элементы списка справа конфликтуют с горизонтальной шапкой. Во втором варианте он сохранен: внимание в первую очередь направленно на заголовок Workspace1.

Р

исунок 2.2. Цветовое акцентирование на главное рабочей зоне

Следует использовать только сознательно подобранные цвета. Это основная ошибка очень большого количества дизайнеров. Часто можно видеть пример, когда дизайнер берет элемент, основываясь на абстрактных представлениях о том, что «подтвердить» должно быть зеленым, а «отменить» — красным. Помимо явной ограниченности такого подхода есть еще и проблема того, что не все понимают, что, если ты использовал определенный зеленый цвет, ты можешь к нему использовать только строго определенный красный. Случайных цветов не бывает вообще: человеческий глаз способен улавливать малейшие отличия и подсознательно всегда воспринимает цветовой диссонанс. Более того, следует помнить, что большинство современных экранов имеют весьма ограниченные цветовые пространства, и диссонансы на них представляются гораздо более грубыми, нежели, скажем, на холсте. Пара примеров подобного отношения (рисунок 2.3.):

Рисунок 2.3. Различные варианты использования цвета в одном и том же приложении.



Р

исунок 2.3. Различные варианты использования цвета в одном и том же приложении.

Обратим внимание на то, что ни один компонент элемента, будь то цвет текста, цвет обводки значков, цвет обводки кнопки, не оставлен без внимания — это создает цельность и гармоничность. Оттенок кнопки совершенно необязательно должен быть зеленым, важно, что если это зеленый, то он не случаен.

Существует несколько ограничений, которые нужно всегда держать в голове:

— Никогда не использовать черный и серый цвет текста на цветном фоне. Это создаёт ощущение грязи. На белом фоне, кстати, тоже далеко не во всех случаях можно использовать чистый черный цвет.

— Не использовать глухой серый для фонов. Если на макете есть хотя бы один цветной элемент, всегда следует окрашивать серые элементы в его оттенок.

— Не использовать правый крайний угол палитры для основных цветов.

Ниже отмечена приемлемая зона выбора четкого цвета, но не бледных оттенков (Рисунок 2.4):

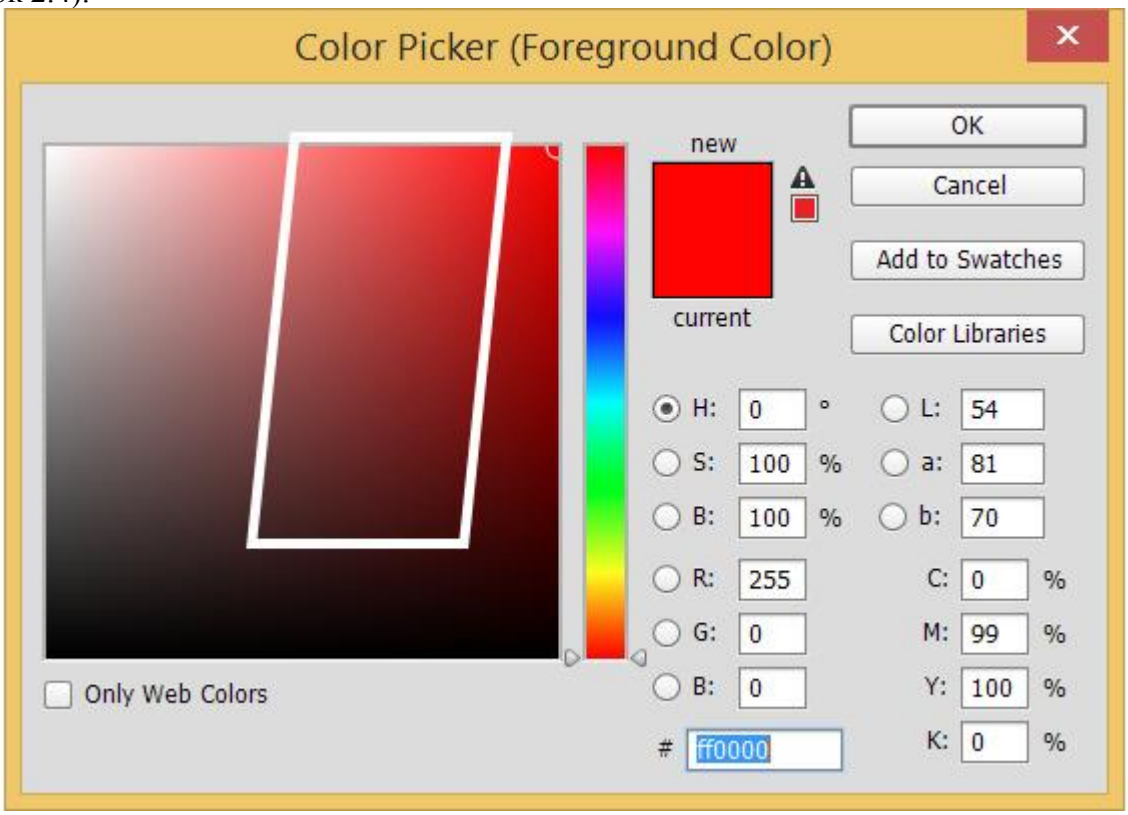


Рисунок 2.4. Зона выбора оптимальных цветов для основных элементов.

Кстати, веб-цвета тоже настоятельно не рекомендуется использовать, особенно учитывая современное развитие браузеров. Веб-цвета — это всего лишь оптимизация значений, они на удивление грубы и плохо соотносятся между собой. Во время разработки палитры интерфейса стоит использовать различные генераторы палитр и цветовых схем. Например это Kuler от компании Adobe или Paletton (Рисунок 2.5.). Разумеется, все вышеперечисленное следует воспринимать с умом, а не в лоб. Для того, чтобы грамотно подобрать практически любой цвет, как правило, приходится подвигать ползунок тона, т.к. чистые относительные цвета обычно недостаточно/чрезмерно контрастны для своих задач.

Рисунок 2.5. Генератор цветовых схем.

Человеческое зрение очень хорошо умеет адаптироваться к условиям различного контраста, например, его статический контраст равен примерно 100:1, а динамический может достигать 1.000.000:1, поэтому не следует бояться его понижения внутри элементов. При этом восприятие контрастности повышается с уменьшением циклов (грубо говоря, разноцветных элементов). В отличие от реального мира, компьютерный интерфейс имеет весьма примитивную структуру контрастов, небольшое количество цветов и типов элементов. Хороший пример грамотно выстроенного контраста — текущая версия социальной сети Facebook (Рисунок 2.6.):

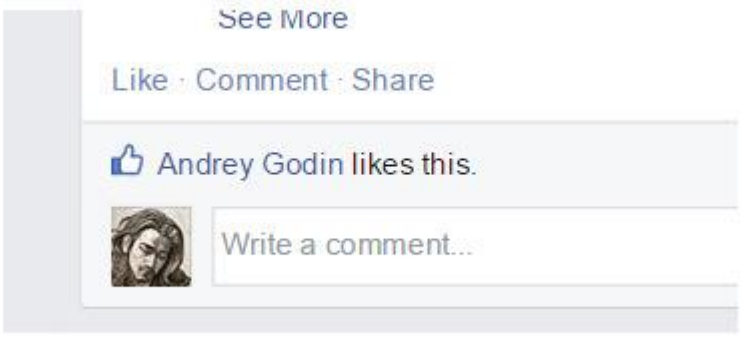


Рисунок 2.6. Цвет и контраст в интерфейсе Facebook.

Обратим внимание на то, что легкие контрасты между областями тем не менее хорошо отделяют их друг от друга. Визуально сохраняется даже наследственность (область комментариев явно принадлежит общей области поста), и это заслуга не только обводки.

Хорошим показателем того, что палитра в интерфейсе удалась, если он создает «на вкус» ощущение единого освещения. Это достигается путем использования естественных контрастов, насыщенных оттенком теней. Например, не рекомендуется

использовать холодное на теплом в качестве акцента. Это противоестественно, в природе такого не бывает. Даже подобранные по цветовым отношениям синий в красном смотрится хуже, чем красный в синем (Рисунок 2.7):

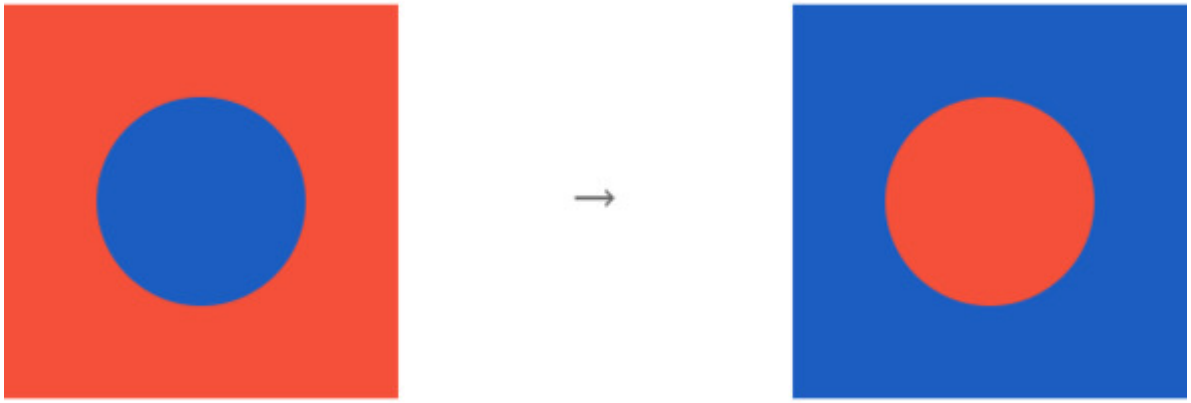


Рисунок 2.7. Использование естественных контрастов

Также следует давать пользователю ясный фокус на одном первостепенном сценарии и делать остальные сценариии явно второстепенными. Отсюда следует хрестоматийное call-to-action. Однако следует понимать, что в 90% случаев call-to-action является кнопкой и предваряется какой-то информативной частью. Это означает что она должна быть легко находима, но не бросаться в глаза первой. Это достигается, например, при помощи яркого, активного цвета в элементе небольшого размера (Рисунки 2.8; 2.9; 2.10)

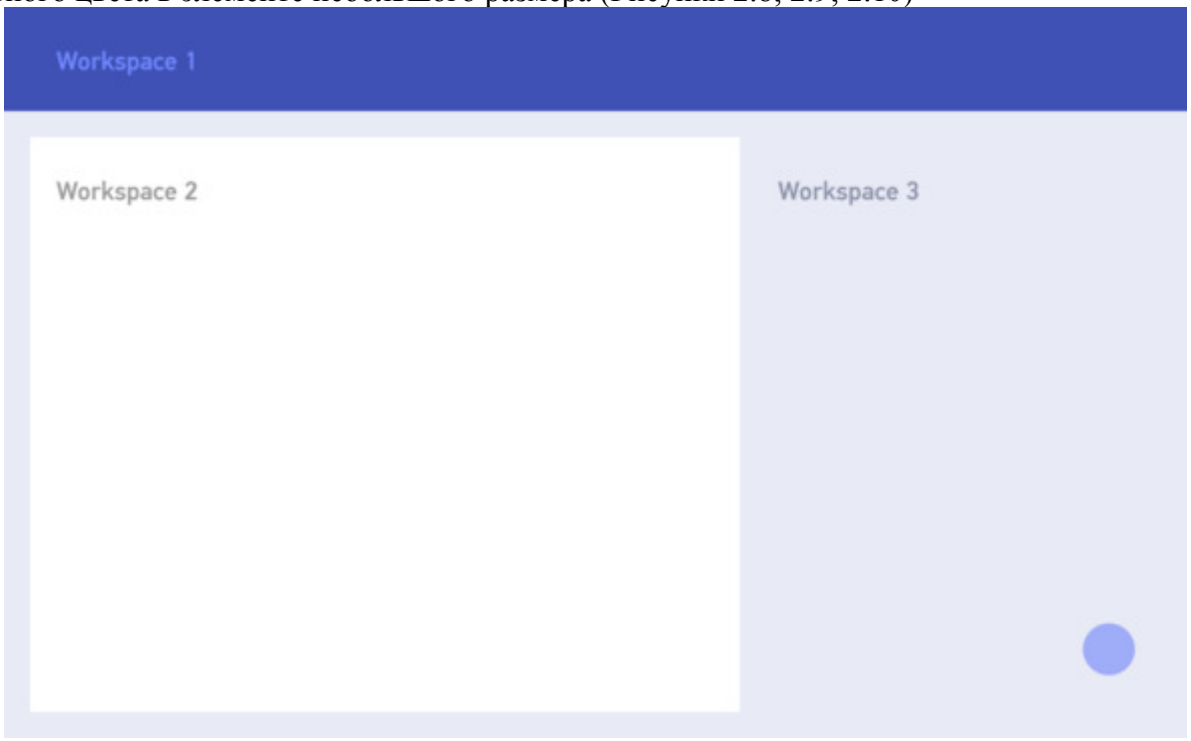


Рисунок 2.8.Call-to-action как часть сценария workspace 2.

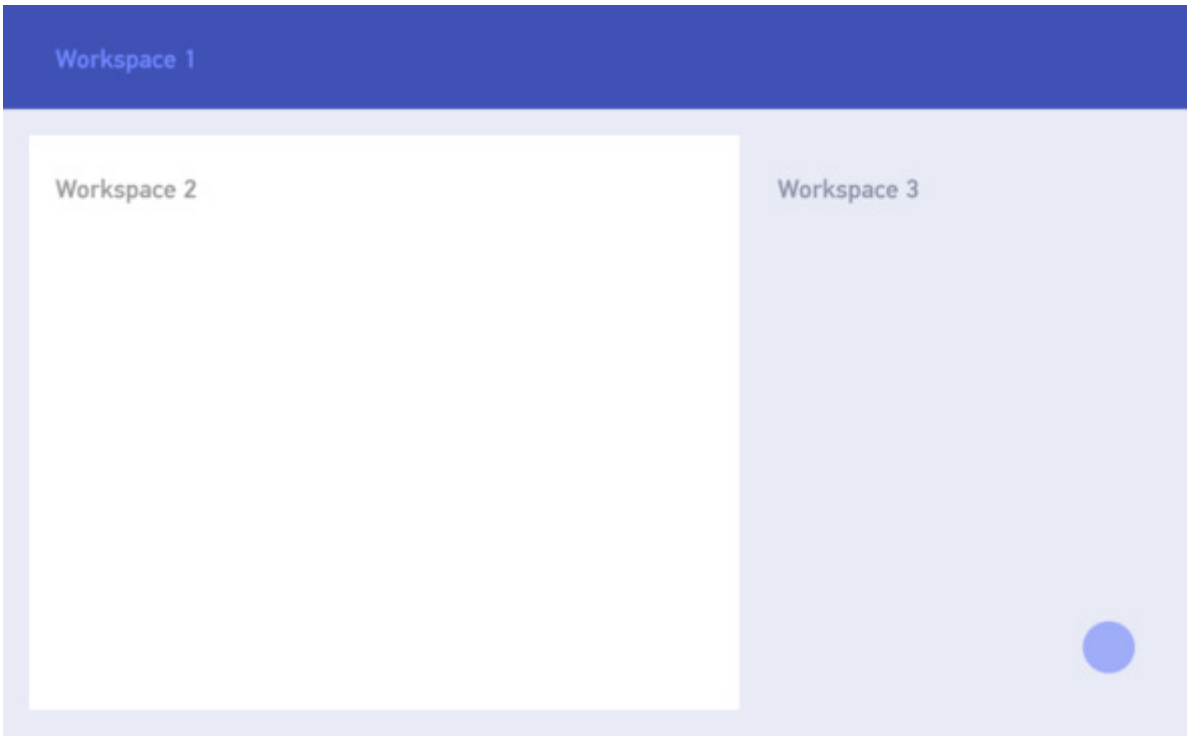


Рисунок 2.9.Call-to-action как часть сценария workspace 3.

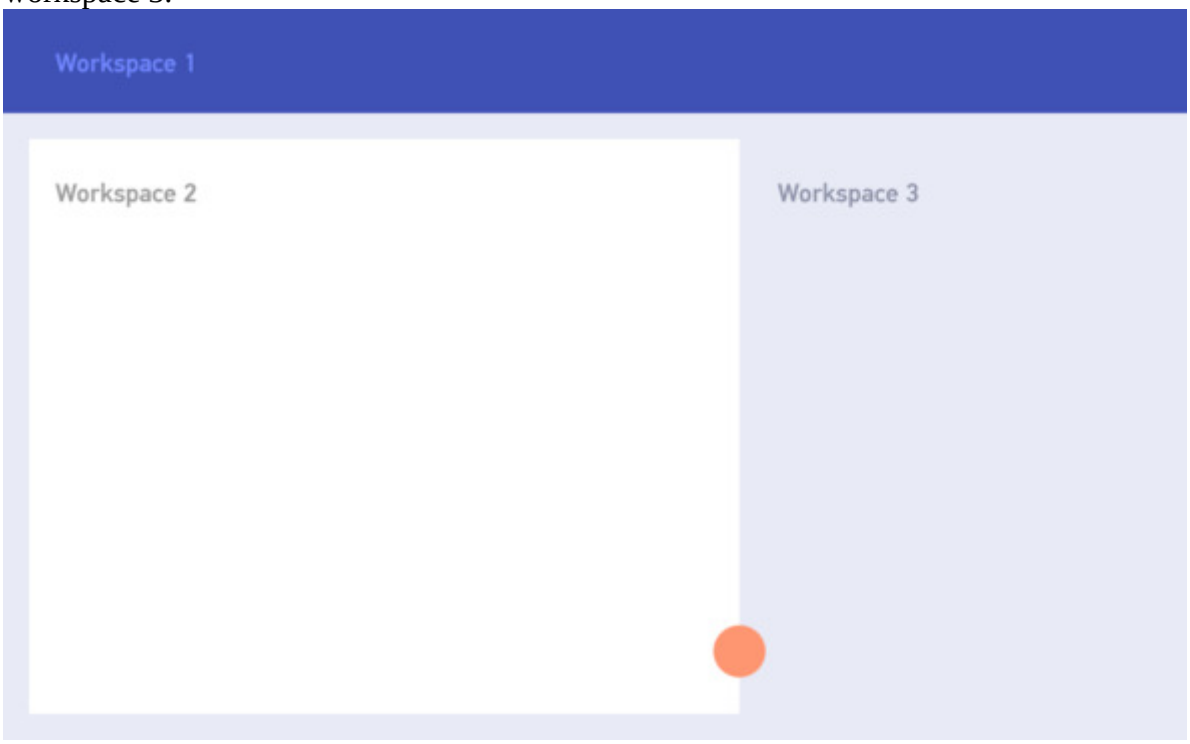


Рисунок 2.10.Call-to-action как отдельный сценарий.

3.Мобильный интерфейс для одной руки

В своей книге Designing Mobile Interfaces (2011) дизайнер Стивен Хубер ввел понятие The Thumb Zone («зона большого пальца») — область экрана, наиболее удобная при использовании телефона одной рукой. С года издания книги средний размер смартфона заметно увеличился, и «мертвая зона» — область, которую сложно достать пальцем одной руки, — также стала больше (Рисунок 3.1):

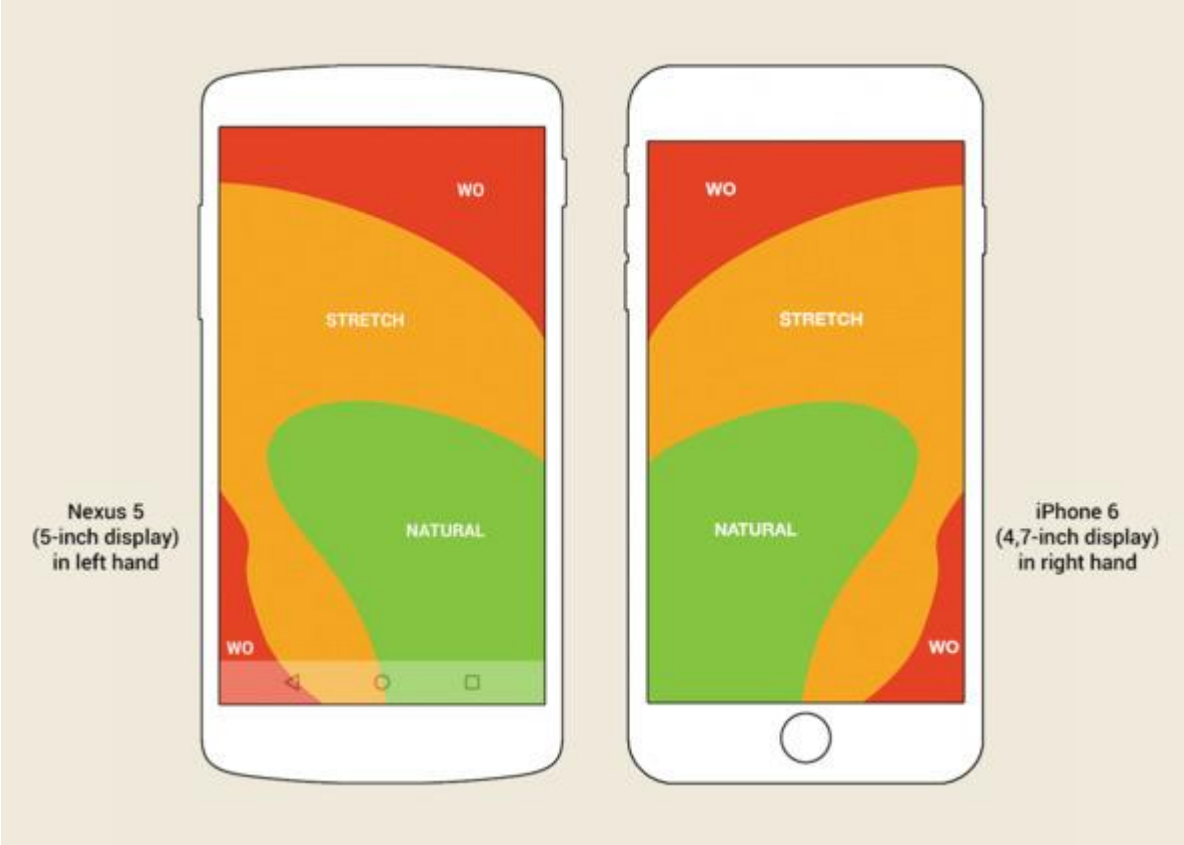


Рисунок 3.1. «Зона большого пальца» для 5–4,7-дюймового экрана.

В «мертвую зону», отмеченную красным, попадают панели инструментов приложений, которые как в Android (App Bar / Primary Toolbar), так и в iOS (Navigation Bar) находятся в верхней части экрана.

Настоящее решение данной проблемы должно не помогать дотянуться до нужной кнопки, а избавить пользователя от необходимости дотягиваться.

В эпицентре «мертвой зоны» iOS стандартно расположена кнопка Back, на телефонах Android в данную область попадает кнопка вызова бокового меню (Navigation Drawer). Однако в большинстве приложений нет необходимости к ней тянуться — достаточно сделать свайп от левого края телефона (Рисунок 3.2):

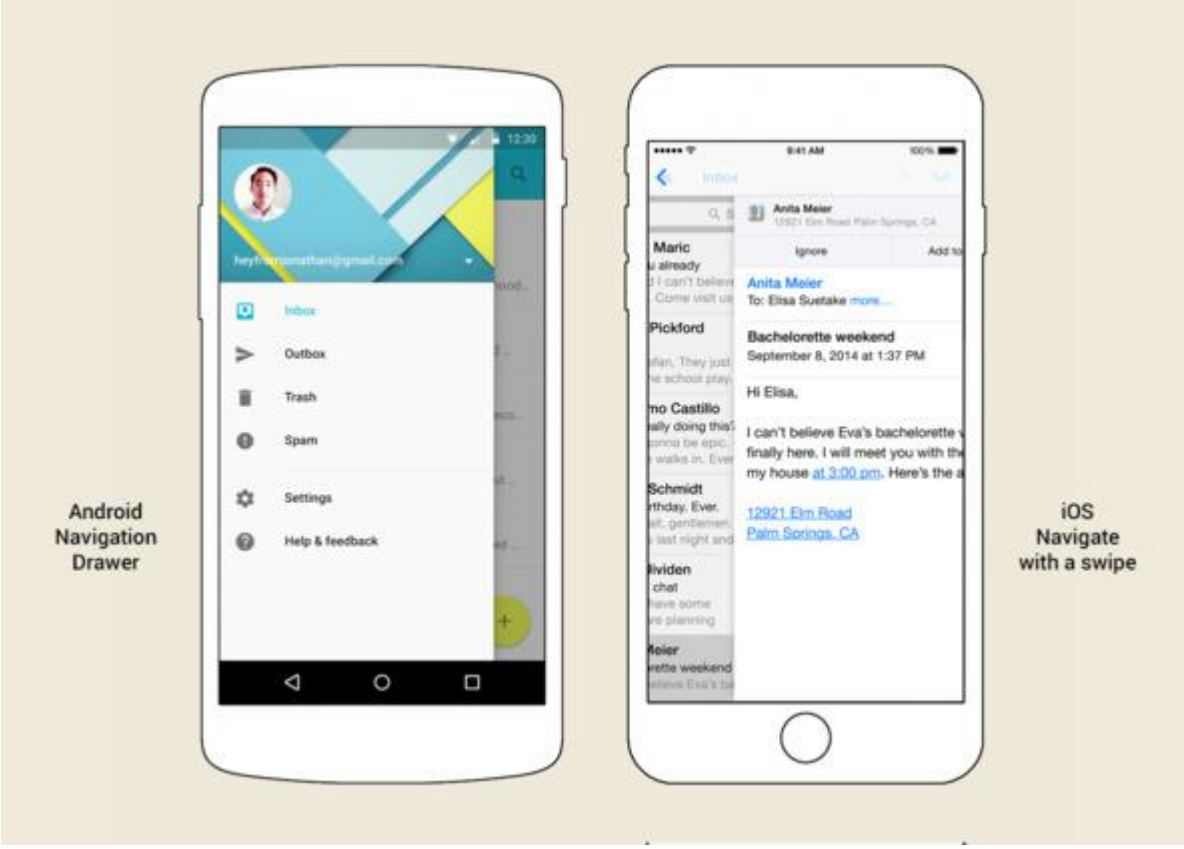


Рисунок 3.2. Боковое (главное) меню (Navigation Drawer) в Android и функция Navigare with a swipe («Навигация смахиванием») в iOS.

Но не бывает правил без исключений. С правой стороны панели инструментов могут находиться кнопки «Готово» или «Отправить», которые ведут к необратимому действию. Такое простое движение как swipe не должно приводить к выполнению необратимых действий.

Использование swipe от левого и правого края телефона для активации функций верхней панели инструментов легко применимо как для iOS, так и для Android. Более того, подобный функционал в отношении левой стороны уже частично реализован на обеих платформах. Такое интуитивно-понятное поведение имеет все шансы стать стандартом мобильных интерфейсов.

Гайдлайны для различных мобильных платформ

Разрабатывая интерфейс, следует учитывать, на какой платформе будет выпускаться данное приложение: каждая платформа имеет свой отличительный стиль (Рисунки 4.1, 4.2, 4.3) которого и следует придерживаться и своем контенте. Можно заметить, что каждая платформа использует особенный шрифт, имеет различные размеры главных элементов интерфейса, по-своему группирует объекты на рабочей области:

Рисунок 4.1 Интерфейс IOS 8

Рисунок 4.2 Интерфейс Windows

Рисунок 4.3 Интерфейс Android

Гайдлайны — это попытка описать набор правил для создания интерфейса приложений, то есть того самого опыта, который пользователи получают взаимодействуя с ОС и ее приложениями. Они выполняют двойную задачу. С одной стороны, использование гайдлайнов при проектировке приложения позволяет сократить время на проектирование, за счет использования готовых решений. С другой позволяет сократить время, которое тратит пользователь на обучение работы с вашим приложением. Но, следует понимать и то, что любой набор готовых решений не совершенен. Гайдлайны это рекомендации, их можно нарушать.

В ходе разработки интерфейсов дизайнеру, рано или поздно придется столкнуться с официальной документацией- ведь именно оттуда будет изыматься информация о структуре элементов. Каждая платформа имеет собственные стандарты на иконки, меню, и бекграунды. Как дизайнер, вы должны не усложнять работу и себе, и программистам: лучше сразу сделать все по стандартам, дабы избежать переделывания в будущем.

Функции устройства

Apple предоставляет возможность использовать функции телефона, такие как: акселерометр, работа с файловой, работа с почтой и др.

Существует сразу готовые решения, которые реализованы в различных фреймворках, таких как: core motion, message и т. д.

Из-за консервативной системы Apple — функционал, предоставляемый приложениям, ограничен. Нет возможности использовать системные возможности, такие как: работа с питанием, работа с файловой системой (приложение может только читать файлы, а изменять может только в своей «песочнице») и т. д.

Акселерометр

Акселерометр, с технической точки зрения, представляет из себя устройство, способное измерять ускорение предмета, которое оно приобретает при смещении относительно своего нулевого положения. Акселерометр применяется как для измерения ускорения в сторону, в которую произошло смещение, так и для измерения ускорения, вызванного силой тяжести Земли. Не работает в вакууме.

Для того, чтобы пользоваться акселерометром, необходимо подключить фреймворк «CoreMotion» и импортировать в «ViewController» класс “ <CoreMotion/CoreMotion. h>». Добавить кнопку (создайте связь с вью), по нажатию на которую будет перемещена в центр экрана. Нужна для того, чтобы отслеживать, как влияет положение устройство на кнопку. Задайте положение устройства — портретная ориентация слева/справа (чтобы приложение не вращалось, портретная т.к. вектор направления ускорения x и y, при вертикальной ориентации, поменяются местами.).

Добавьте следующий список свойств в интерфейс класса вью:

1. @property (strong, nonatomic) CMMotionManager *motionManager;
2. @property (assign, nonatomic) CMAcceleration acceleration;
3. @property (strong, nonatomic) NSOperationQueue *queue;
4. @property (strong, nonatomic) NSDate *lastUpdateTime;
5. @property (assign, nonatomic) CGPoint currentPoint;
6. @property (assign, nonatomic) CGPoint previousPoint;
7. @property (assign, nonatomic) CGFloat buttonXVelocity;
8. @property (assign, nonatomic) CGFloat buttonYVelocity;

1. Исходя из названия, понятно его предназначение.

2. Структура, содержащая 3 вектора направления ускорения (x,y,z).

3. Данный класс позволяет создавать очередь из операций, которые выполняются по порядку и приоритету.

4. Класс дата, позволяет узнавать дату.

5. Текущая точка. 6. Предыдущая точка.

7,8 Вектор направления ускорения по x и y.

Далее, в методе «viewDidLoad» добавьте следующий код:

_motionManager = [[CMMotionManager alloc] init]; // ваша точка входа в сервис движений

_motionManager.deviceMotionUpdateInterval = 1/60; // как часто будем получать обновления, это означает, что 60 раз получим данные за одну секунду

_currentPoint = _buttonCenter.frame. origin; // текущая точка

_queue = [[NSOperationQueue alloc] init]; // Класс, который позволяет создавать очередь из операций и выполнять их в порядке очереди, а также приоритета

[_motionManager startAccelerometerUpdatesToQueue:_queue withHandler:

^ (CMAccelerometerData *accelerometerData, NSError *error) {

[self setAcceleration:accelerometerData.acceleration];

[self performSelectorOnMainThread:@selector (update) withObject: nil waitUntilDone: NO];

}]; // В этом методе, мы вносим в очередь операций, затем задаем обработчик в виде блока, который принимает параметры — данные акселерометра и описание ошибки, вызываем метод «update», он будет вызываться 60 раз в секунду.

Блок — это логически сгруппированный набор идущих подряд инструкций в исходном коде программы. Блоки служат для ограничения области видимости переменных и функций, а также позволяют обращаться к блоку инструкций как к единой инструкции. Блоки являются основой парадигмы структурного программирования.

Реализуем метод «update»:

— (void) update {

NSTimeInterval secondsSinceLastDraw = — ([_lastUpdateTime timeIntervalSinceNow]); // так как, интервал обновления данных может изменяться, мы должны учитывать время прошедшее с предыдущего обновления до нового.

_buttonYVelocity = _buttonYVelocity — (_acceleration. x * secondsSinceLastDraw); //

_buttonXVelocity = _buttonXVelocity — (_acceleration. y * secondsSinceLastDraw); // вектор направления с учетом времени между последним и текущем вызовом акселерометра, этот вектор и будет являться скоростью

CGFloat xDelta = secondsSinceLastDraw * _buttonXVelocity * 500; //

CGFloat yDelta = secondsSinceLastDraw * _buttonYVelocity * 500; // высчитываем приращение, где 500 — это расстояние, чем выше будет это значение, тем более дерганным будет движение кнопки

_currentPoint = CGPointMake (_currentPoint. x + xDelta, _currentPoint. y + yDelta); // определяем новую точку

[self moveButton]; // передвигаем кнопку

_lastUpdateTime = [NSDate date]; // задаем время последнего обновления

}

Реализуем метод «moveButton»:

— (void) moveButton {

[self collisionWithBoundaries]; // проверяем, не вылетела ли кнопка за пределы экрана

_previousPoint = _currentPoint; // задаем предыдущую точку

CGRect frameNew = [_buttonCenter frame]; // создаем новые «границы» для кнопки

frameNew. origin. x = _currentPoint. x;

frameNew. origin. y = _currentPoint. y;

[_buttonCenter setFrame: frameNew];

}

Теперь наша кнопка зависит от положения устройства в пространстве. Но у нас кнопка вылетает за пределы экрана. Исправим это, реализуем метод «collisionWithBoundaries» с отскоком от края экрана:

— (void) collisionWithBoundaries {

if (_currentPoint. x <0) {// не вылетела ли кнопка за пределы экрана

_currentPoint. x = 0; // то возвращаем в пределы экрана

_buttonXVelocity = — (_buttonXVelocity / 2.0); // меняя вектор на положительный

} // учтите, что «якорная» точка расположена в левой части по середине кнопки

if (_currentPoint. y <0) {// по аналогии с верхним условием

_currentPoint. y = 0;

_buttonYVelocity = — (_buttonYVelocity / 2.0);

}

if (_currentPoint. x> self.view.bounds.size. width — _buttonCenter.frame.size. width) {// т.к. «якорная» точка слева по середине

_currentPoint. x = self.view.bounds.size. width — _buttonCenter.frame.size. width;

_buttonXVelocity = — (_buttonXVelocity / 2.0); // задаем отрицательный вектор

}

if (_currentPoint. y> self.view.bounds.size. height — _buttonCenter.frame.size. height) {// по аналогии с предыдущим условием

_currentPoint. y = self.view.bounds.size. height — _buttonCenter.frame.size. height;

_buttonYVelocity = — (_buttonYVelocity / 2.0);

}

}

Обратите внимание, что мы обращаемся к «границам» самой кнопки, а не к ее содержимому, следовательно, необходимо учитывать пространство между границами кнопки и текстом.

Теперь наша кнопка может отскакивать от края экрана. Для удобства, добавьте следующий код в действие кнопки, который будет выполнятся при нажатие на кнопку, чтобы она появлялась в центре экрана:

— (IBAction) buttonCenter: (id) sender {

CGSize viewSize = self.view.bounds.size; // размеры экрана

[_buttonCenter setFrame: CGRectMake (viewSize. width/2, viewSize. height/2, _buttonCenter.frame.size. width, _buttonCenter.frame.size. height)]; // задаем позицию, не меняя размеров кнопки

_currentPoint = CGPointMake(_buttonCenter.frame. origin. x, _buttonCenter.frame. origin. y); // записываем новую позицию

}

Теперь, при нажатие на кнопку, она будет появляется в центре экрана.

Работа с камерой

Еще одна доступная функция iOS — это камера. С ее помощью можно делать снимки и видео в вашем приложение.

Все что нам необходимо для работы с камерой — это подключить протокол <UIImagePickerControllerDelegate> и <UINavigationControllerDelegate>. В данной главе, создадим приложение, в котором будет возможность создать фотографии и видео, затем вставить их в «UIImageView», а также использовать будет верхнюю панель кнопок от «Navigation Controller», поэтому добавьте его в проект. И добавьте на панель кнопку — «Bar Button Item» (рис 3.1), затем добавьте в интерфейс класса как «action».

Добавьте «Image View» и свяжите его с контроллером. Теперь осталось осуществить переход в камеру и обратно.

Для использования камеры добавьте следующий код в методе кнопки:

— (IBAction) takePhoto: (id) sender {

UIImagePickerController *picker = [[UIImagePickerController alloc] init];

picker.delegate = self;

picker.sourceType = UIImagePickerControllerSourceTypeCamera; // указываем назначение нашего объекта

[self presentViewController: picker animated: YES completion: nil]; // осуществляем анимированный переход, третий параметр принимает блок, который будет исполняться после завершения перехода

}

После того, как сделали фотографию будет вызван метод, в котором возвращается фотография. Дополним его тем, что будем выводить полученное изображение на экран:

— (void) imagePickerController: (UIImagePickerController *) picker didFinishPickingMediaWithInfo: (NSDictionary *) info {

UIImage *foto = [info objectForKey:@«UIImagePickerControllerOriginalImage»]; // получаем фото по ключу

_fotoView.image = foto;

UIImageWriteToSavedPhotosAlbum (Foto, nil, nil, nil); // сохраняем изображение

[picker dismissViewControllerAnimated: YES completion: nil]; // возвращаемся на вью контроллер, убирая и уничтожая представление камеры

}

Теперь, с помощью приложения можно делать фотографии и сохранять их.

Видеозапись. Добавим в наш проект объект класса «UISegmentedControl», в котором будет две кнопки — это запись и воспроизведение сделанной записи. Затем свяжите (property и action) с контроллером, чтобы это сделать, необходимо выделить его как показано на рисунке, а затем уже связывать (рис. 3.1).

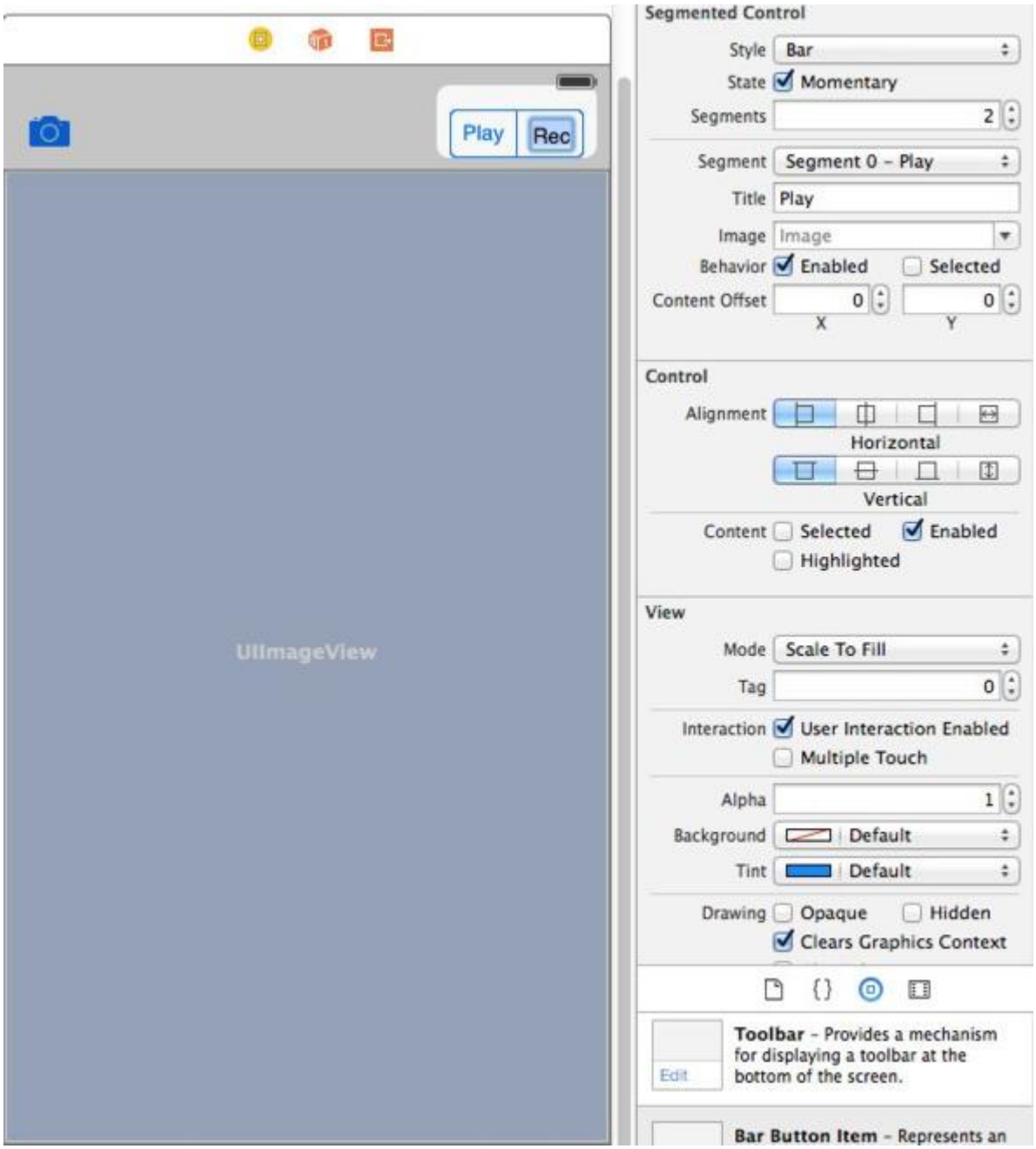


Рисунок 3.1 — Настройка «SegmentedControl»

Далее добавить два сегмента, и убрать галочку с selected на каждом сегменте. Добавить галочку в momentary, чтобы сегмент не был подсвечен синим, после того как на него нажали (рис. 3.1).

Для работы с видеокамерой требуется два фреймворка — это «MediaPlayer», «MobileCoreServices», «AssetsLibrary». Первый необходим для воспроизведения видео. Второй нужен для доступа к «kUTTypeMovie» (медиа формат, содержащий как видео, так и аудио, обеспечивает доступ к видеокамере), также и для доступа к другим сервисам телефона. Третий необходимо для сохранения видео в галерею устройства.

В первую очередь необходимо импортировать список классов:

#import <MobileCoreServices/MobileCoreServices. h>

#import <MediaPlayer/MediaPlayer. h>

#import <AssetsLibrary/AssetsLibrary. h>

Затем добавьте одну переменную класса «NSURL», хранящая ссылку на файл, может хранить как путь к файлу на устройстве, так и ссылку из интернета.

Реализуем методы, вызываемый по нажатию на один из сегментов:

— (IBAction) video: (id) sender {

```
if (_videoSeg.selectedSegmentIndex == 0 && videoURL) {// создаем сам плеер, проверяем, чтобы ссылка не была пустой
MPMoviePlayerViewController *player = [[MPMoviePlayerViewController alloc] initWithContentURL: videoURL]; // Создаем плеер с ссылкой на записанный файл
/*player.view.frame = CGRectMake (0, 64, 375, 200);
[self.view addSubview:player.view]; Плеер можно добавить на сам вью */
[self presentMoviePlayerViewControllerAnimated: player];
} else if (_videoSeg.selectedSegmentIndex == 1) {// создаем контроллер камеры
UIImagePickerController *picker = [[UIImagePickerController alloc] init];
picker.delegate = self;
picker.sourceType = UIImagePickerControllerSourceTypeCamera;
picker.mediaTypes = [NSArray arrayWithObjects: (NSString *) kUTTypeMovie, nil]; // указываем тип съемки, приводим к типу стринг
[self presentViewController: picker animated: YES completion: nil];
}
}
```

Теперь осталось дополнить метод, который вызывается после закрытия камеры:

```
— (void) imagePickerController: (UIImagePickerController *) picker didFinishPickingMediaWithInfo: (NSDictionary *) info
{
UIImage *Foto = [info valueForKey:@«UIImagePickerControllerEditedImage»];
if (Foto) {
_imageView.image = Foto;
UIImageWriteToSavedPhotosAlbum (Foto, nil, nil, nil); // сохраняем изображение
}
if (videoURL) {
ALAssetsLibrary* library = [[ALAssetsLibrary alloc] init];
[library writeVideoAtPathToSavedPhotosAlbum: videoURL
completionBlock:^ (NSURL *assetURL, NSError *error) {
NSLog (@«Video Saved»);
}]; // сохраняем видео на устройстве}
[picker dismissViewControllerAnimated: YES completion: NULL];
}
```

В этом методе получаем ссылку на видеозапись, затем сохраняем ее в галерею.

С помощью нашего приложение, можно делать переход в камеру и снимать видео, делать фото. Также сохраняются файлы на устройстве.

Основные объекты UIKit

Мы с вами уже познакомились с парой таких объектов — это надпись, кнопка, вью контроллер. Теперь рассмотрим получше основной фреймворкXcode.

Рассмотрим сначала два важных понятия, а затем приступим к дальнейшему изучению объектов.

IBOutlet–это важные instance-переменные (внутренние), которые ссылаются надругие (внешние) объекты. Outlets ввашем коде могут быть типизированы слабо или сильно. Слаботипизированные outlets (тип которых — id) могут бытьпривязаны к любому объекту, а сильно типизированные outlets могут быть привязаны только к томуобъекту, класс которого совпадает с типом outlet.

Action — это сообщения, которые в ответ на определенные события один объект посылает другому объекту.

С двумя этими понятиями будем сталкиваться часто.

PickerView

Существует множество программ (в том числе и нативных), которые используют UIPickerView. Стандартный таймер который поставляется вместе с iOS использует UIPickerView для установки времени. По своей сути UIPickerView — это та же таблица, только с анимацией прокрутки.

Создадим проект, затем добавим пиккер вью на ViewController и свяжем с ним (рис. 1.1.1).

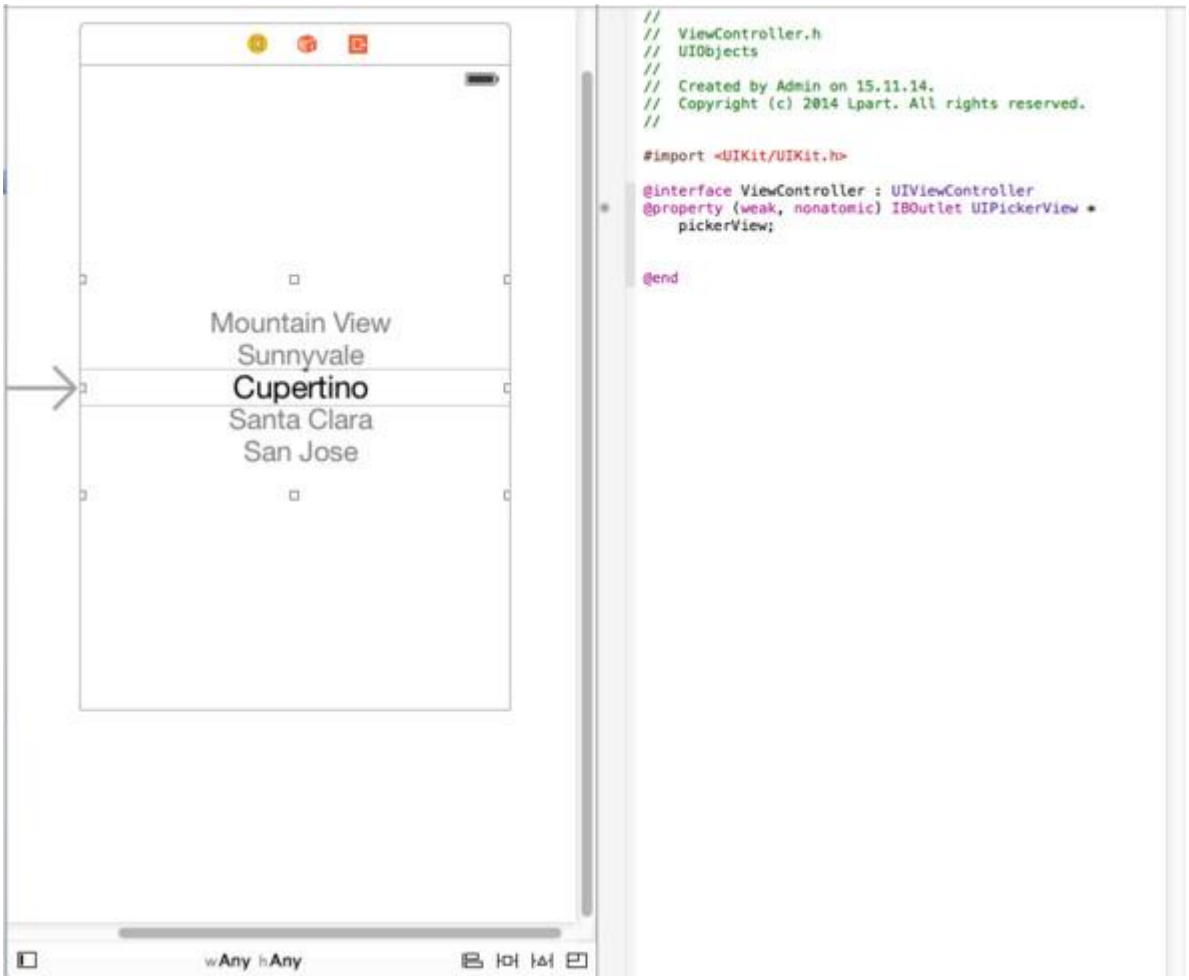


Рисунок 1.1.1 — Добавление PickerView

Теперь нам необходимо подключить протоколы «UIPickerViewDataSource, UIPickerViewDelegate"к классу ViewControllerв его представление. Первый протокол содержит обязательные методы для реализация — это количество строк и компонентов в пиккер вью. Второй протокол содержит такие методы как: didSelectRow, titleForRow, и другие. Первый протокол нужен для успешной инициализации пиккер вью, а второй протокол содержит необязательные методы, которые вызываются в соответствие с выполненными действияминад пиккер вью. Должна получится следующая конструкция кода:

@interfaceViewController: UIViewController <UIPickerViewDataSource, UIPickerViewDelegate>

@property (weak, nonatomic) IBOutletUIPickerView *pickerView;

Необходимо еще указать делегирование PickerView, чтобы он инициализировался при загрузке ViewController, для этого зажмите ПКМ на пиккер вью и протяните стрелку до первого элемента, затем, в появившемся списке выберете пункт «delegate» (рис. 1.1.2). Делегирование — это, когда объект для предоставления определённого набора функциональности полагается на другой объект.

Рисунок 1.1.2 — Делегирование полномочий

Теперь выведем данные из массива со строками в пиккер вью. Нам необходимо создать массив и наполнить его строками. Затемреализоватьметоды «numberOfRowsandnumberOfComponents», иметоды «didSelectRow"и «titleForRow». В итоге мы должны получить пиккер вью, наполненный некими строками, и он должен вызывать метод «didSelectRow», в котором выводится в консоль строка, на которой остановился пиккер вью, но мы получаем только индекс выбранной строки, нам нужно по индексу достать из массива значение. Кодреализации:

```
— (void) viewDidLoad {
[superviewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
arrayStrings = [NSArray arrayWithObjects:@«Hello», @«People», @«Hello», @«World», nil];
}
— (void) didReceiveMemoryWarning {
[superdidReceiveMemoryWarning];
// Dispose of any resources that can be recreated.
}
#pragma mark — — — Picker View — — —
— (NSInteger) numberOfComponentsInPickerView: (UIPickerView *) pickerView {
return1; // будемвозвращатьодинкомпонент
}
— (NSInteger) pickerView: (UIPickerView *) pickerViewnumberOfRowsInComponent: (NSInteger) component {
```

returnarrayStrings.count; // кол-во строк = кол-во элементов массива

}

— (NSString *) pickerView: (UIPickerView *) pickerViewtitleForRow: (NSInteger) row forComponent: (NSInteger) component {

return [arrayStringsobjectAtIndex: row]; //вернем строку из массива с индексом текущей инициализируемой строки

}

— (void) pickerView: (UIPickerView *) pickerViewdidSelectRow: (NSInteger) row inComponent: (NSInteger) component {

NSLog (@"%@», [arrayStringsobjectAtIndex: row]);

}

Результат работы на рисунке 1.1.3. Мы разобрали основы работы с пиккер вью, так же можно добавлять строки с различными атрибутами, можно создавать вью из нескольких компонентов.

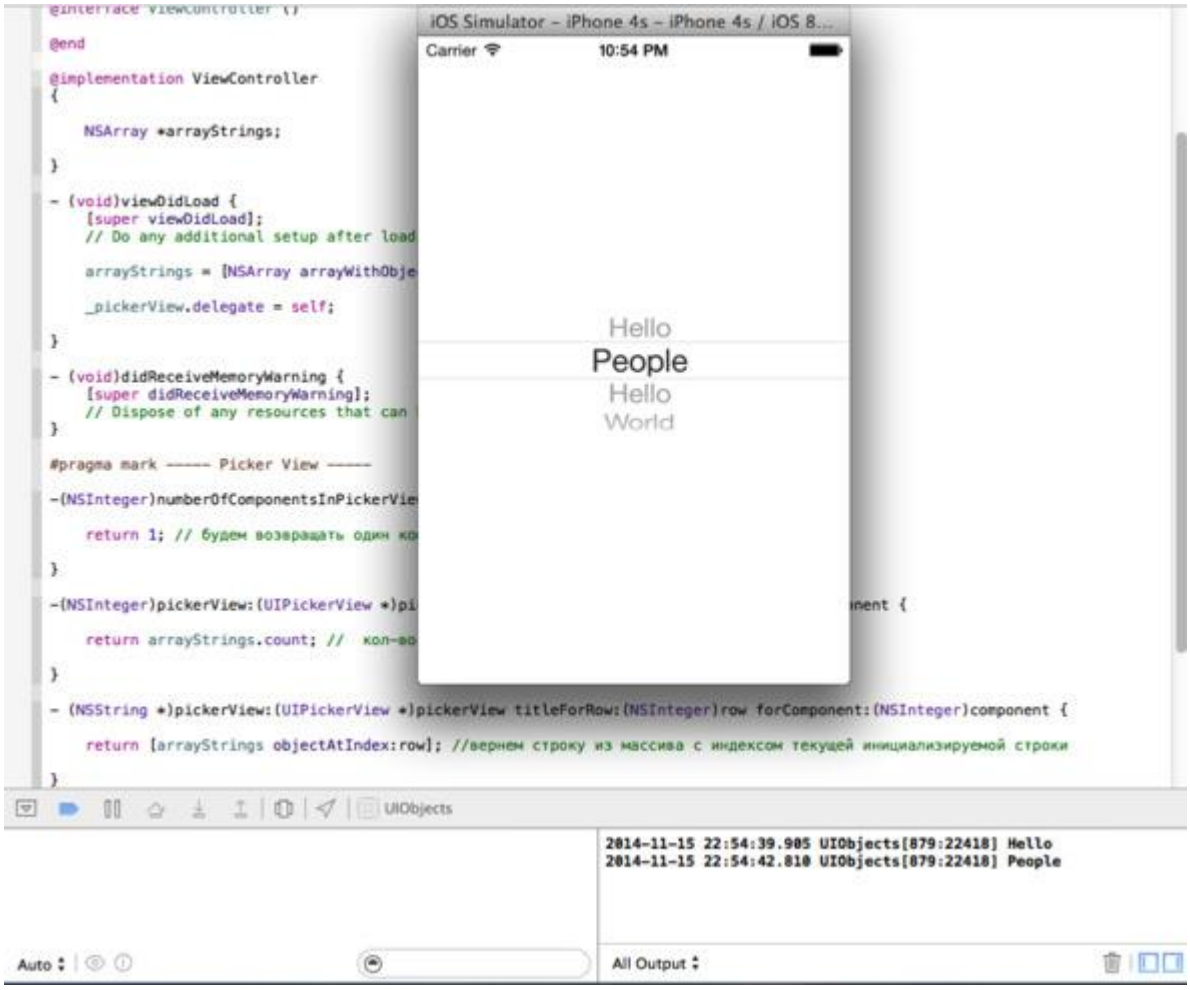


Рисунок 1.1.3 — Пример работы приложения

1.2TableView и TextField

Таблицы являются базой для большинства списков выбора на iOS. Почта, список контактов, последние звонки — все эти приложения используют возможности класса UITableView. Чаще всего таблицы используются вместе с UINavigationController, который обеспечивает навигацию между таблицей и детальным представлением элемента.

Рассмотрим простой пример работы с TableView. Для этого, добавим таблицу на вью контроллер и свяжем с ним. И нам необходимо реализовать три метода: «numberOfSectionsInTableView», «numberOfRowsInSection», «cellForRowAtIndexPath». Должен получится такой код:

@implementationViewController

```
{
NSArray *arrayStrings;
}
— (void) viewDidLoad {
[superviewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
arrayStrings = [NSArrayarrayWithObjects:@«Hello», @«People», @«Hello», @«World», nil];
_simpleTable.delegate = self; //
_simpleTable. dataSource = self; // такимобразом, будувызваныметодыэтихдвухпротоколов
}
— (void) didReceiveMemoryWarning {
[superdidReceiveMemoryWarning];
// Dispose of any resources that can be recreated.
}
#pragma mark — — — Table View — — —
— (NSInteger) numberOfSectionsInTableView: (UITableView *) tableView {
return1; // кол-восекций
}
— (NSInteger) tableView: (UITableView *) tableViewnumberOfRowsInSection: (NSInteger) section {
returnarrayStrings.count; // кол-во строк = кол-ву элементов массива
}
— (UITableViewCell *) tableView: (UITableView *) tableViewcellForRowAtIndexPath: (NSIndexPath *) indexPath {
NSString *cellIdentifier = @«Cell»;
UITableViewCell *cell = [tableViewdequeueReusableCellWithIdentifier: cellIdentifier];
if (cell == nil) {
cell = [[UITableViewCellalloc] initWithStyle: UITableViewCellStyleDefault
reuseIdentifier: cellIdentifier];
}
cell. textLabel. text = [arrayStringsobjectAtIndex:indexPath.row];
cell. textLabel. textColor = [UIColorblackColor];
returncell;
}
```

Разберемся с последним методом, который вызывается столько, сколько всего строк есть в таблице. Сначала мы создаем идентификатор, после создаем ячейку из той, что есть в таблице по этому идентификатору, если таковой нет, то создаем ячейку с нуля. Дальше записываем строку из массива по номеру строки в ячейку, затем назначаем цвет текста, возвращаем ячейку.

Теперь будем редактировать ячейки. Для этого нам необходимо указать таблицы, что она изменяема:

```
— (void) viewDidLoad {
[superviewDidLoad];
arrayStrings = [NSArrayarrayWithObjects:
@«Hello», @«People», @«Hello», @«World», nil];
_simpleTable.delegate = self;
_simpleTable. dataSource = self;
_simpleTable. editing = YES;
}
```

Теперь появилась кнопка удалить (рис. 1.2.1).

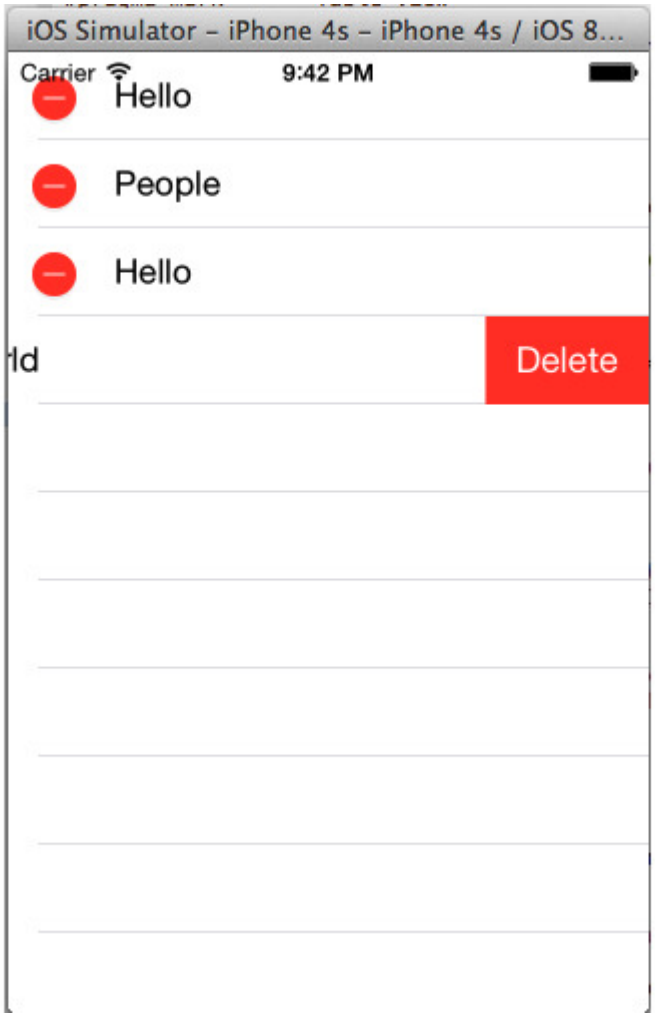


Рисунок 1.2.1 — Удаление ячейки

Но она не будет работать, так как мы не реализовали метод, которые вызывается этой кнопкой. Нужно нам поменять еще и наш массив на изменяемый массив, чтобы мы могли добавлять в него или удалять объекты. Необходимо добавить следующую конструкцию кода в класс:

— (void) tableView: (UITableView*) tableViewcommitEditingStyle: (UITableViewCellEditingStyle) editingStyleforRowAtIndexPath: (NSIndexPath *) indexPath {
if (editingStyle == UITableViewCellEditingStyleDelete) {
[arrayStringsremoveObjectAtIndex:indexPath.row];
[tableViewdeleteRowsAtIndexPaths: [NSArrayarrayWithObject: indexPath] withRowAnimation: UITableViewRowAnimationFade];
}
}

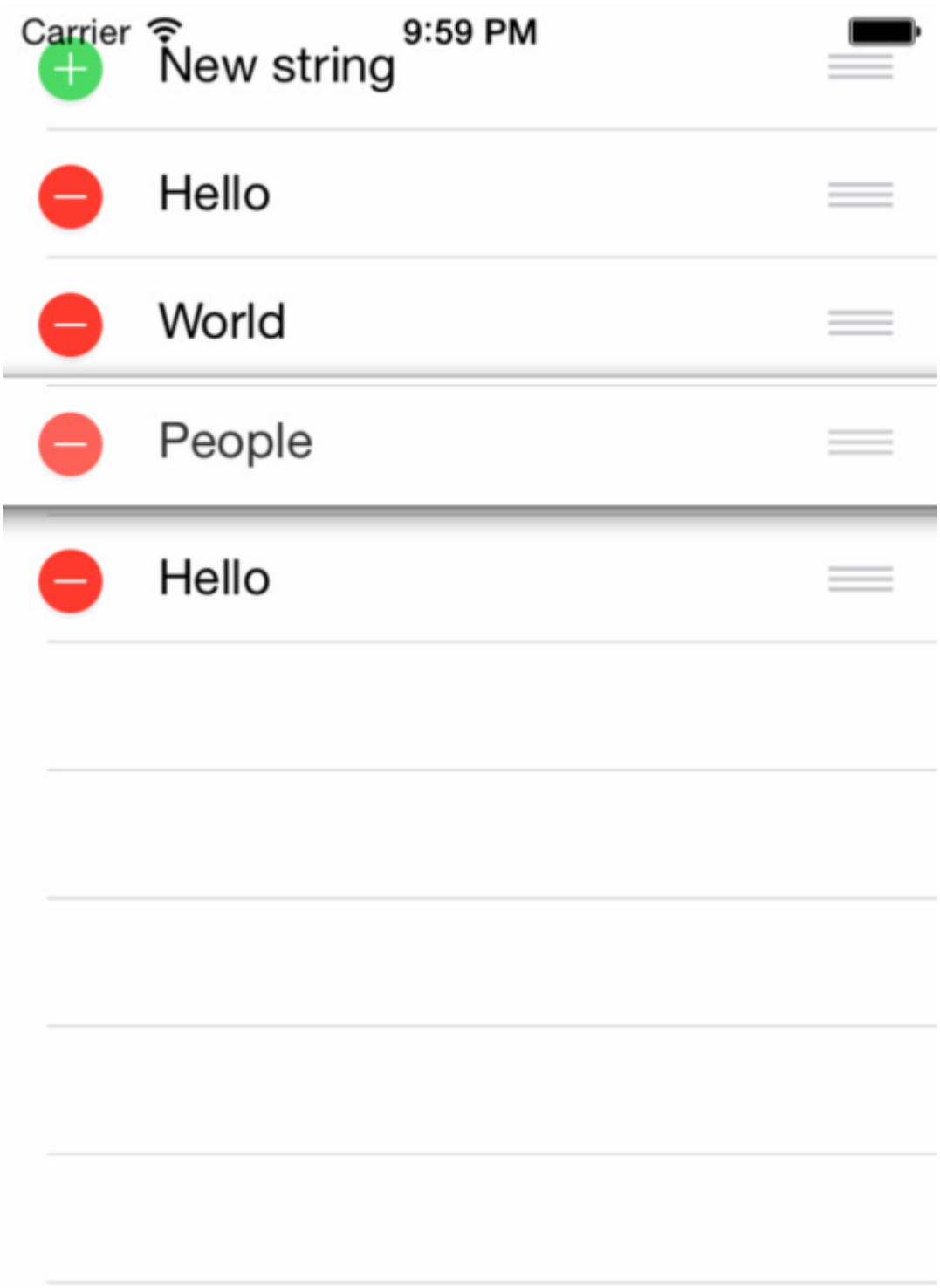
Теперь мы можем удалять ячейки и объекты из массива. Добавим еще возможность вставления значений и перемещение ячеек.

Необходимо реализовать три метода, это: «canMoveRows», «moveRows», «editingStyleForRows». Требуется еще доработать метод «commitEditingStyle», в него необходимо добавить проверку на вставление ячейки. И тогда код примет следующий вид:

— (void) tableView: (UITableView *) tableViewcommitEditingStyle: (UITableViewCellEditingStyle) editingStyleforRowAtIndexPath: (NSIndexPath *) indexPath {
if (editingStyle == UITableViewCellEditingStyleDelete) {
[arrayStringsremoveObjectAtIndex:indexPath.row];
[tableViewdeleteRowsAtIndexPaths: [NSArrayarrayWithObject: indexPath] withRowAnimation: UITableViewRowAnimationFade];
}

```
if (editingStyle == UITableViewCellEditingStyleInsert) {
[arrayStringsinsertObject:@«New string"atIndex:indexPath.row];
[tableViewreloadData];
}
}
— (BOOL) tableView: (UITableView *) tableViewcanMoveRowAtIndexPath: (NSIndexPath *) indexPath {
returnYES;
}
— (void) tableView: (UITableView *) tableViewmoveRowAtIndexPath: (NSIndexPath *) sourceIndexPathtoIndexPath: (NSIndexPath *) destinationIndexPath {
[arrayStringsexchangeObjectAtIndex:sourceIndexPath.rowwithObjectAtIndex:destinationIndexPath.row];
NSLog (@"%@», arrayStrings);
}
— (UITableViewCellEditingStyle) tableView: (UITableView *) tableVieweditingStyleForRowAtIndexPath: (NSIndexPath *) indexPath {
if (indexPath.row == 0) {
returnUITableViewCellEditingStyleInsert;
}
else {
returnUITableViewCellEditingStyleDelete;
}
}
```

Теперь можно удалять, вставлять строки, передвигать их (рис. 1.2.2).



Рисунок

1.2.2 — Редактирование ячеек

Настраиваемая ячейка

Разберем тот случай, когда нам нужно ячейка, в которой есть различные объекты UIKit, в большинстве случаев — надписи, кнопки.

Чтобы добавлять объекты в ячейку, мы должны добавить саму ячейку в нашу таблицу и связать с вью. Далее, необходимо создать собственный класс, наследуя UITableViewCell, и присвоить нашей ячейке, нужно для того, чтобы связать объекты с ячейкой, и задать идентификатор ячейке. Добавим надпись на ячейку и свяжем с ней. Последний шаг — это изменить код в методе «cellForRows…» :

— (UITableViewCell *) tableView: (UITableView *) tableViewcellForRowAtIndexPath: (NSIndexPath *) indexPath {

NSString *cellIdentifier = @«extendCell»;

TableViewCell *cell = [tableViewdequeueReusableCellWithIdentifier: cellIdentifier];

if (cell == nil) {

cell = [[TableViewCellalloc] initWithStyle: UITableViewCellStyleDefault

reuseIdentifier: cellIdentifier];

}

cell. textLabel. textColor = [UIColorblackColor];

cell.mainTitle. text = [arrayStringsobjectAtIndex:indexPath.row];

returncell;

}

Таким образом, мы создаем объект «cell» класса «TableViewCell», в котором находится propertyс надписью. Результат программы на рисунке 1.2.3.

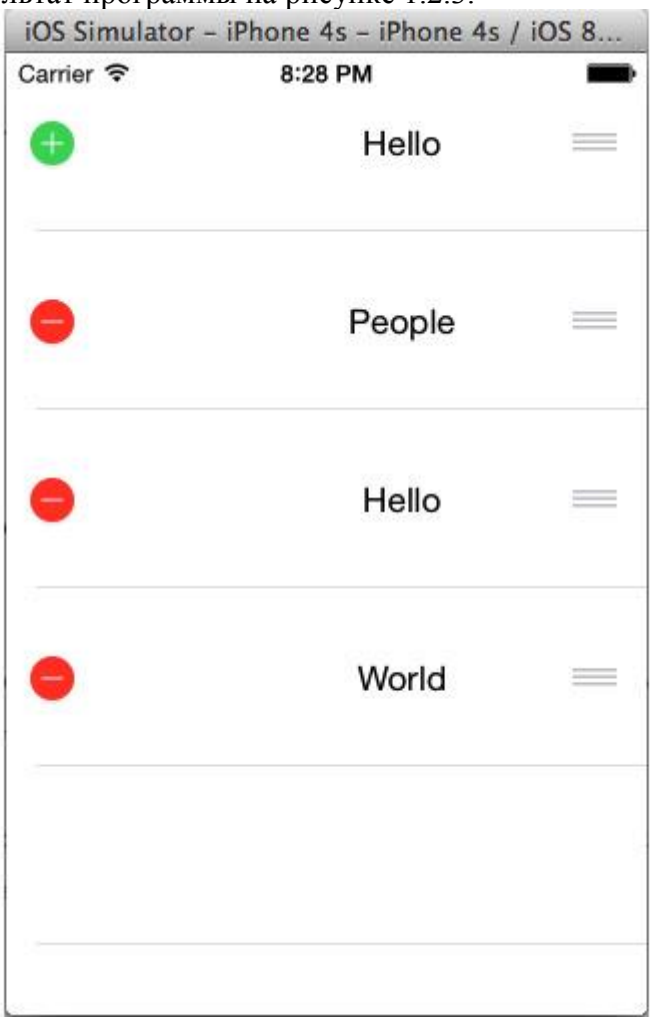


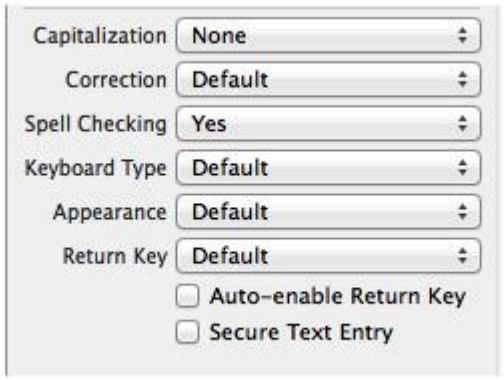
Рисунок 1.2.3 — Настраиваемая ячейка

Несмотря на то, что надпись было помещена по центру, она была смещена из кнопки редактирования. Это можно исправить, выставив констрейны надписи, чтобы она всегда была посередине. Следующей темой будет — Navigationcontroller, в ней будем снова работать с нашей таблицей.

TextField

Это объект, позволяющий вводить, редактировать, принимать текст с клавиатуры.

Создадим проект и добавим на вью контроллер текстовое поле. Разберемся с его основными атрибутами (рис. 1.2.4).



Р

исунок 1.2.4 — Основные атрибуты

1) Capitalization– имеет несколько параметров: каждое слово с большой буквы, каждое предложение с большой буквы, все буквы — заглавные.

2) Correction — авто-исправление вводимого текста.

3) SpellChecking — проверка на орфографию.

4) KeyboardType–список определенных типов клавиатур: ASCII, URL, NumbarPad, PhonePadи т. д.

5) Appearance — определяет цвет заднего фона клавиатуры

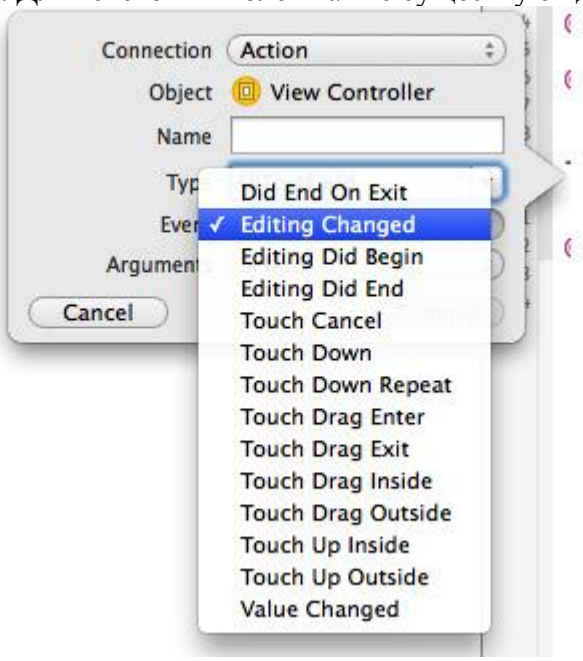
6) ReturnKey — типкнопкиподтверждения: Done, Cancel, Search, Yahoo ит. д.

7) Auto-enable — когда текстовое поле пустое, то кнопка неактивна и наоборот.

8) SecureTextEntry — заменяется символы на точки.

Остальные атрибуты рассмотрим в процессе изучения.

Теперь для наглядности добавим два текстовых поля и две надписи. Необходимо связать их через «IBOutletCollection». Создается одно свойство для каждого объекта, таким образом получается массив объектов. Для текстовых полей также существуют действия (рис. 1.2.5).



Р

исунок 1.2.5 — События TextField

Открываем список Eventи выбираем «EditingChanged». Такой метод будет вызываться каждый раз, как только будет изменен текст (добавлен/удален). В этот же метод добавим второе текстовое поле. И получается так, как показано на рисунке 1.2.6.

```
//
//  ViewController.h
//  testTextField
//
//  Created by Admin on 08/01/15.
//  Copyright (c) 2015 Lpart. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController <UITextFieldDelegate>


@property (strong, nonatomic) IBOutletCollection(UITextField) NSArray *textFields;

@property (strong, nonatomic) IBOutletCollection(UILabel) NSArray *labelWithInputText;


- (IBAction)actionInputingText:(UITextField *)sender;


@end
```

Рисунок 1.2.6 — Список свойств

Будем выводить написанный текст в надписи, но чтобы выводить в надпись напротив, требуется ее определить, будем делать это через тэг.

В методе «viewDidLoad» напишите следующий код:

— (void) viewDidLoad {
[superviewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
inti = 1;
for (UITextField *textFieldin_textFields) {// перебираемкаждыйэлементвмассиве
textField. tag = i;
i++;
}
i = 1;
for (UILabel *label in_labelWithInputText) {
label. tag = i;
i++;
}
}

А в нашем методе, который вызывается после каждого изменения:

— (IBAction) actionInputingText: (UITextField *) sender {
for (UILabel *label in_labelWithInputText) {
if (label. tag == sender. tag) {// если надпись напротив текстового поля
label. text = sender. text;
}
}
}

Но как можете заметить, кнопка возврата не работает. Так как необходимо реализовать для нее код. Чтобы она работала также необходимо делегирование полномочий для UITextField, и подключение протокола, в котором находятся необязательный методы, но позволяющие сильно расширить функционал. В интерфейсе класса подключите протокол <UITextFieldDelegate>. Вернитесь в класс, реализующий вью контроллер, и добавьте:

— (void) viewDidLoad {
[superviewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
inti = 1;
for (UITextField *textFieldin_textFields) {// перебираемкаждыйэлементвмассиве
textField. tag = i;

textField.delegate = self;
i++;
}
……
}
Далее реализуйте метод из протокола «textFieldShouldReturn»:
— (BOOL) textFieldShouldReturn: (UITextField *) textField {
NSLog (@"%i», textField. tag); // отображаем
[textFieldresignFirstResponder]; // сворачиваетклавиатуру
returnYES;
}
При каждом нажатие на клавишу подтверждения, будет вызван этот метод.

Рассмотрим еще один важный метод «shouldChangeCharactersInRange», но для начала, необходимо удалить «actionInputingText» и удалить связи с этим методом.

Добавьтетакойкод:

— (BOOL) textField: (UITextField *) textFieldshouldChangeCharactersInRange: (NSRange) range replacementString: (NSString *) string {
NSLog (@«Last text = %@», textField. text);
NSLog (@«Replace in range = %@», NSStringFromRange (range));
NSString *newString = [textField. textstringByReplacingCharactersInRange: rangewithString: string]; // создаемновуюстрокуизстарой, вставляявстаруюстрокувведённыйтекствзаданномдиапазоне.
NSLog (@«New string = %@», newString);
returnYES;
}

Этот метод принимает текстовое поле, диапазон в котором будет введена новая строка, и сама новая строка.

Усложним немного проект, чтобы в надписи выводились имя и дата рождения (упрощенный вариант). Понадобится проверить, если ли в строке ненужные нам символы, так для имени — это должны быть только буквы, а для даты рождения — только цифры.

Appleразработала класс под названием «NSCharacterSet» и «NSMutableCharacterSet». Они обеспечивают представление символов Юникод. С помощью этого представления можно фильтровать, разделять, удалять строки. В нашем случае, мы будем разделять строку на компоненты с помощью этого класса. Существует еще и просто «NSSet». Этот набор представляет собой неупорядоченную коллекцию объектов. Вы можете использовать наборы как альтернативу массивам, когда порядок элементов не имеет значения и выполнение проверки. Хотя массивы упорядочены, проверка их на членство медленнее, чем проверканабора.

Создадим метод, который будет возвращать булево значение, принимать строку и целое число:

— (BOOL) editLabelWithText: (NSString *) string tag: (NSInteger) tag {
UILabel *labelWithText = [_labelWithInputTextobjectAtIndex: tag — 1]; // получаемнадпись, т.к. первыйэлементначинаетсяс 0, атегначинаетсяс 1, то «-1».
NSIntegermaxLength = 0;
NSCharacterSet *validSet;
switch (tag) {// принимаетцелочисленныйаргумент
case1:
validSet = [[NSCharacterSetletterCharacterSet] invertedSet]; // создаемнаборнебуквенныхсимволов
maxLength = 15;
break;
case2:

validSet = [[NSCharacterSetdecimalDigitCharacterSet] invertedSet]; // создаемнаборсимволов, несодержащийцифр

maxLength = 8;

break;

default:

break;

}

NSArray *components = [string componentsSeparatedByCharactersInSet: validSet]; // разбиваемстроку, есливстречаютсязаданныесимволысимволы

if(!(components.count> 1) &&string. length <= maxLength) {// если не встречаются заданные символы, и удовлетворяет длина, то выводим, иначе игнорируем ввод

labelWithText. text = string;

} else {

returnNO;

}

returnYES;

}

Возвращаем булево значение затем, чтобы знать, можно ли вводить символ или нет.

Таким образом, мы ограничили ввод. Также при попытке вставки ненужных символов, вставка не произойдет.

Navigation&TabBarcontroller

UINavigationController

Обеспечивает навигацию и анимированные переходы между контроллерами представлений (чаще всего с таблицами, контроллер навигации обеспечивает переход от краткого отображения данных в таблице к детальному представлению элемента).

Вернемся к проекту с нашей таблицей. Нам понадобится еще:

— Один дополнительный вью контроллер, с присвоенным ему идентификатором. Вы можете увидеть, что есть два типа идентификаторов — storyboardи restoration. Первый указывается как уникальное значение, второй как уникальное значение, по которому можно воссоздать объект;

— Класс для этого вью;

— Надпись, в которую мы будем выводить значение выбранной ячейки;

— Сам собою, необходимо подключить UINavigationController, предварительно выбрав вью с таблицей (рис. 1.3.1);

— Добавить метод «didSelectRow» в класс «ViewController».

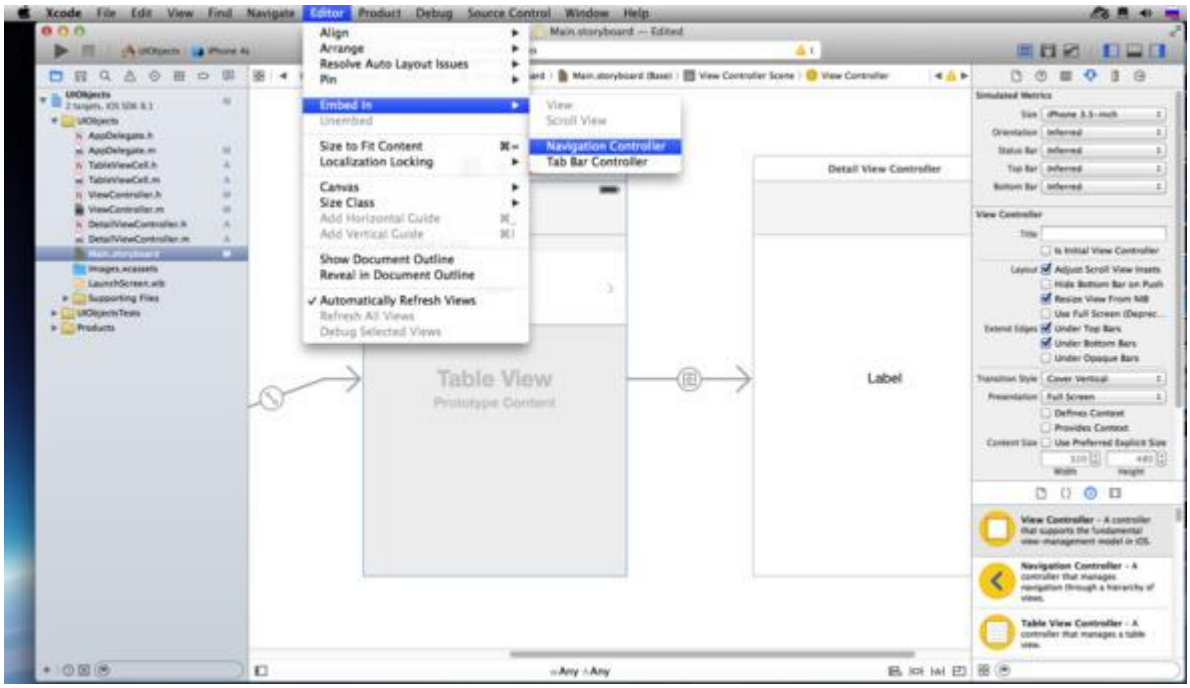


Рисунок 1.3.1 — Подключение NavigationController

Теперь мы установим связь между двумя вью, кнопкой перехода будет служить наша ячейка, связь устанавливается ПКМ (рис. 1.3.2).

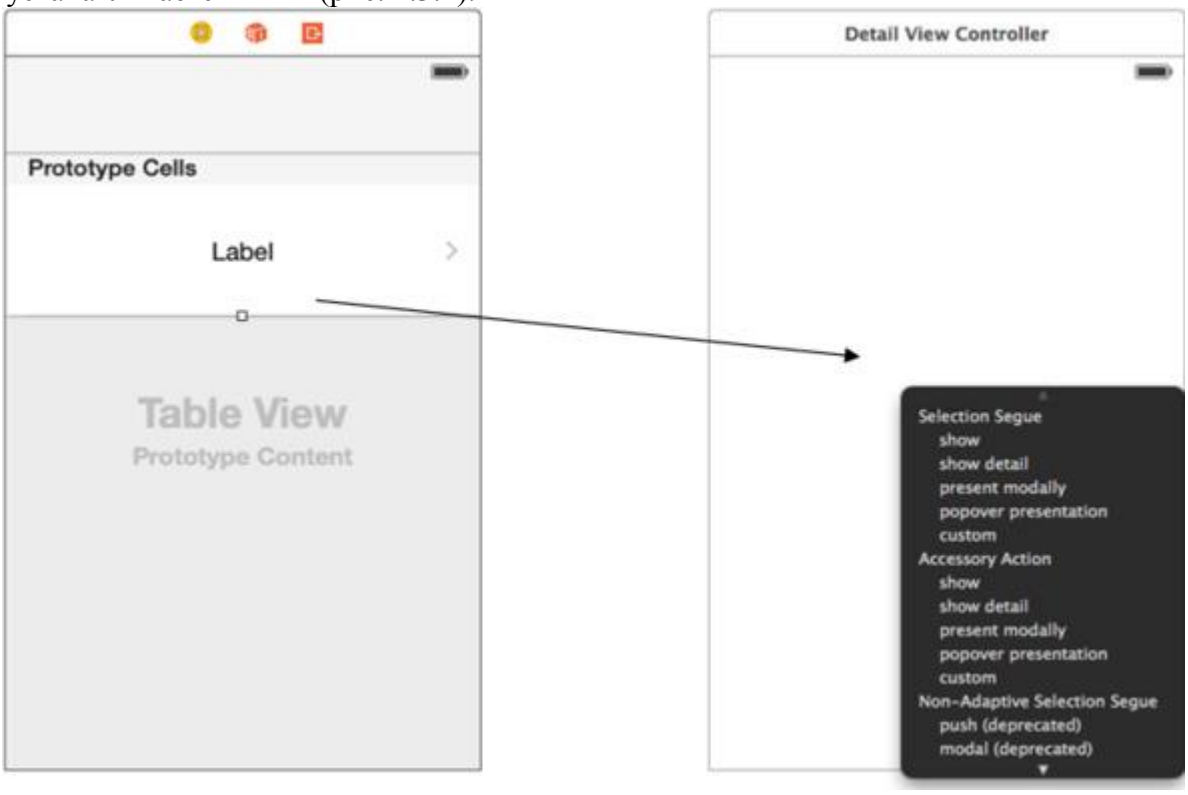


Рисунок 1.3.2 — Установление связи

Появится меню, где выбираются анимации перехода для ячейки и кнопки. Внизу можно заметить пункты, которые устарели, их использовать не рекомендуется, т.к. в новых версиях iOSмогут не поддерживаться, и приложение «упадет». Мы выбрали «show».

Теперь нам необходимо передать данные. Делать будем это через «NSUserDefaults».

Это есть файл с настройками, в который можно добавлять, читать, удалять данные. Он сохраняет эти данные в постоянной памяти. Можно добавить, но не удалить из них данные. Вернемся в класс «ViewController» и реализуем метод в"didSelectRow», в котором будет использовать «NSUserDefaults»:

— (void) tableView: (UITableView *) tableViewdidSelectRowAtIndexPath: (NSIndexPath *) indexPath {

NSUserDefaults *userDefaults = [NSUserDefaultsstandardUserDefaults];

[userDefaultssetObject:[arrayStringsobjectAtIndex:indexPath.row] forKey:@«valueRow»];

}

А в классе второго вью контроллера нам нужно получить значение из файла настроек:

— (void) viewDidLoad {

[superviewDidLoad];

// Do any additional setup after loading the view.

NSUserDefaults *userDefault = [NSUserDefaultsstandardUserDefaults];

NSString *valueRow = [userDefaultvalueForKey:@«valueRow»];

_detailInfo. text = valueRow;

}

Во втором вью, сверху появилась кнопка назад, которая выгружает из стека последнее принятое значение, тем сам возвращаем нам предыдущий вью.

Полученный результат на рисунках 1.3.3 и 1.3.4.

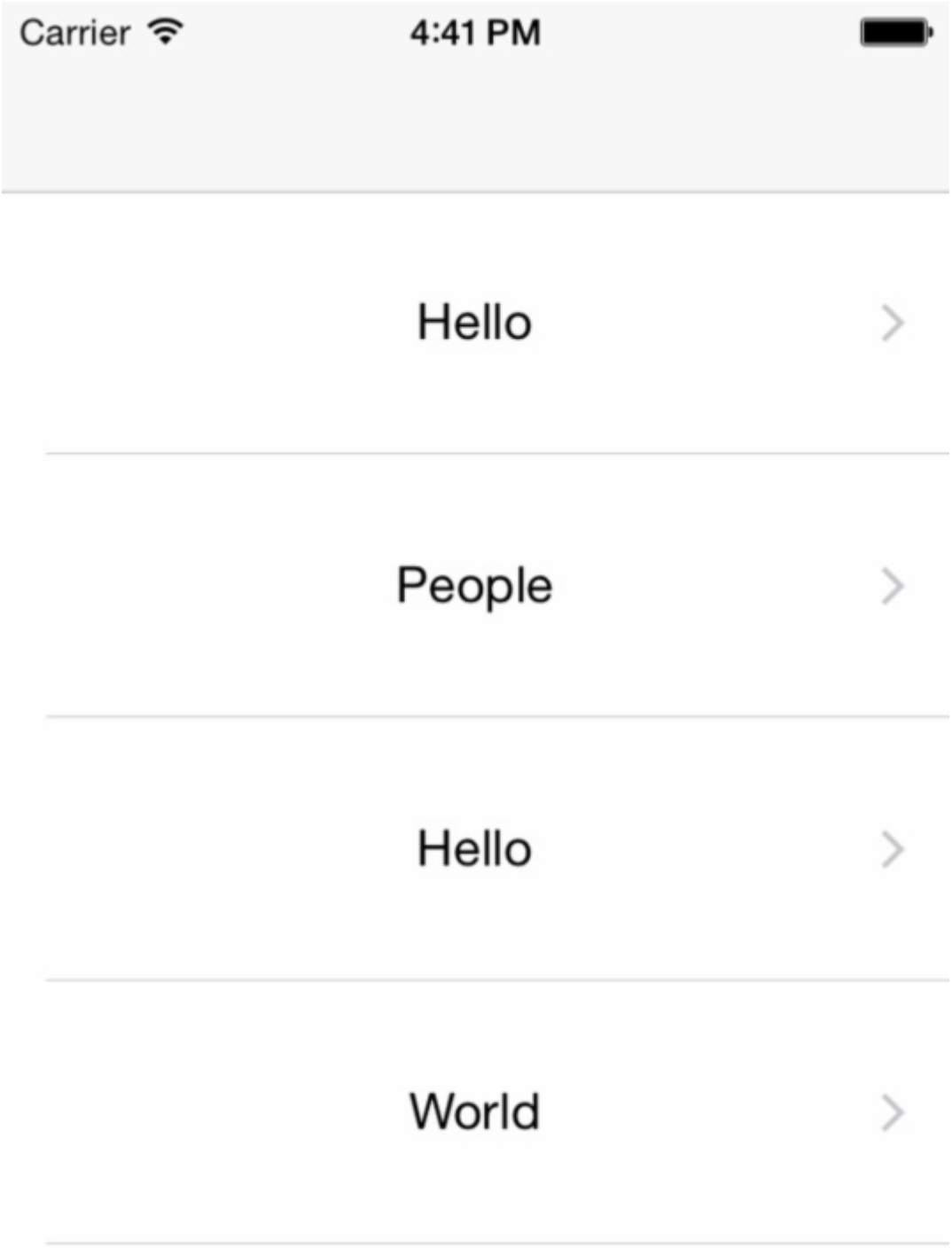


Рисунок 1.3.3 —

Экран №1 Рисунок 1.3.4 — Экран №2

TabBarcontroller

Это еще один способ навигации между вью контроллерами. Только при данном способе, приложение просто переключается между вью, не выполняя заново инициализации.

Уберем navigationcontrollerиз нашего проекта и, вместо него, добавим TabBarcontroller, предварительно выбрав вью. И добавить связь между вторым вью и TabBar, выбрав «viewcontrollers», т.к. остальные пункты не подходят (рис. 1.3.5).

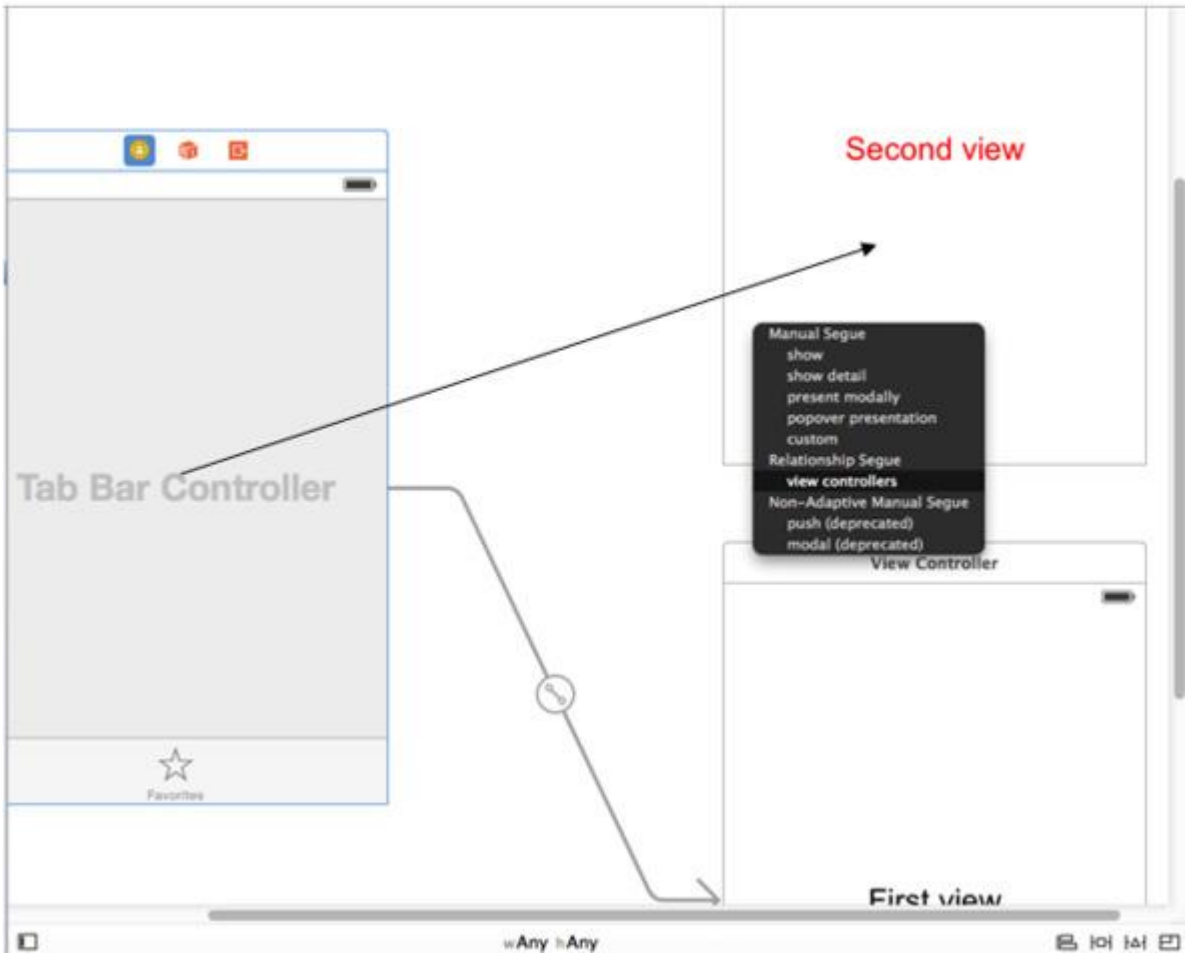


Рисунок 1.3.5 — TabBar

Полученный результат на рисунках 1.3.6, 1.3.7.

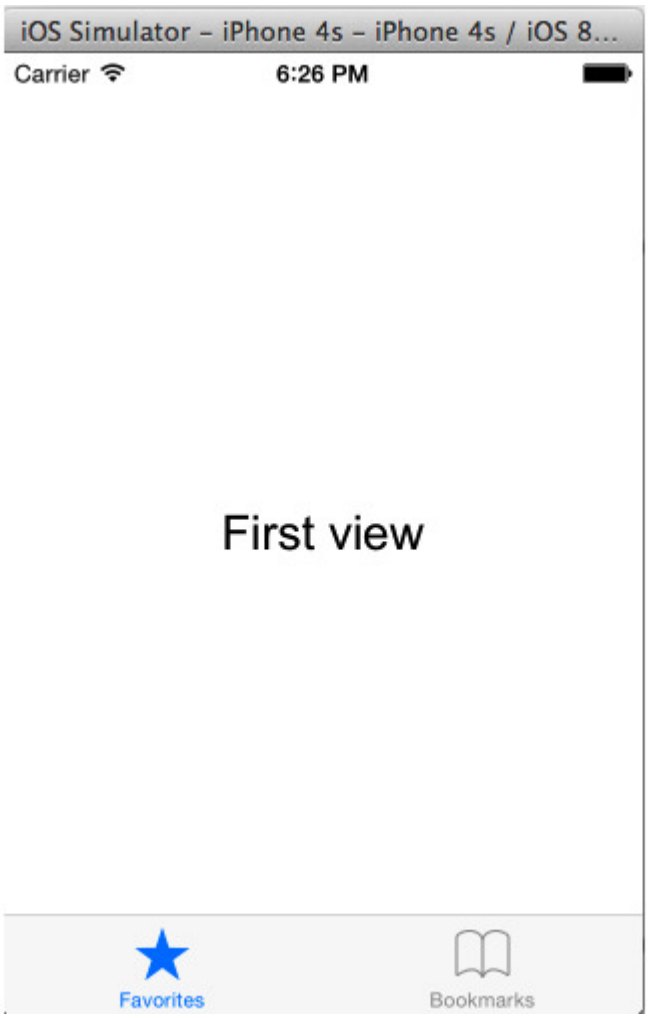


Рисунок 1.3.6 — Первый вью

Рисунок 1.3.7 — Второйвью

GestureRecognizer

UIGestureRecognizer — абстрактный класс, который представляет готовые классы распознавания жестов. Существует несколько готовых классов для определения жестов: нажатие (тап), вращение, долгое нажатие, проведение пальцем (свайп), и т. д.

Добавим распознавание жестов в наш проект. Удалим все объекты из сториборд. Добавим два вью контроллера. Из библиотеки объектов добавьте тап на первый вью и свайп на второй вью. Реализуем переход с первого вью на второй, для этого нам надо связать тап со вторым вью (рис. 1.4.1).

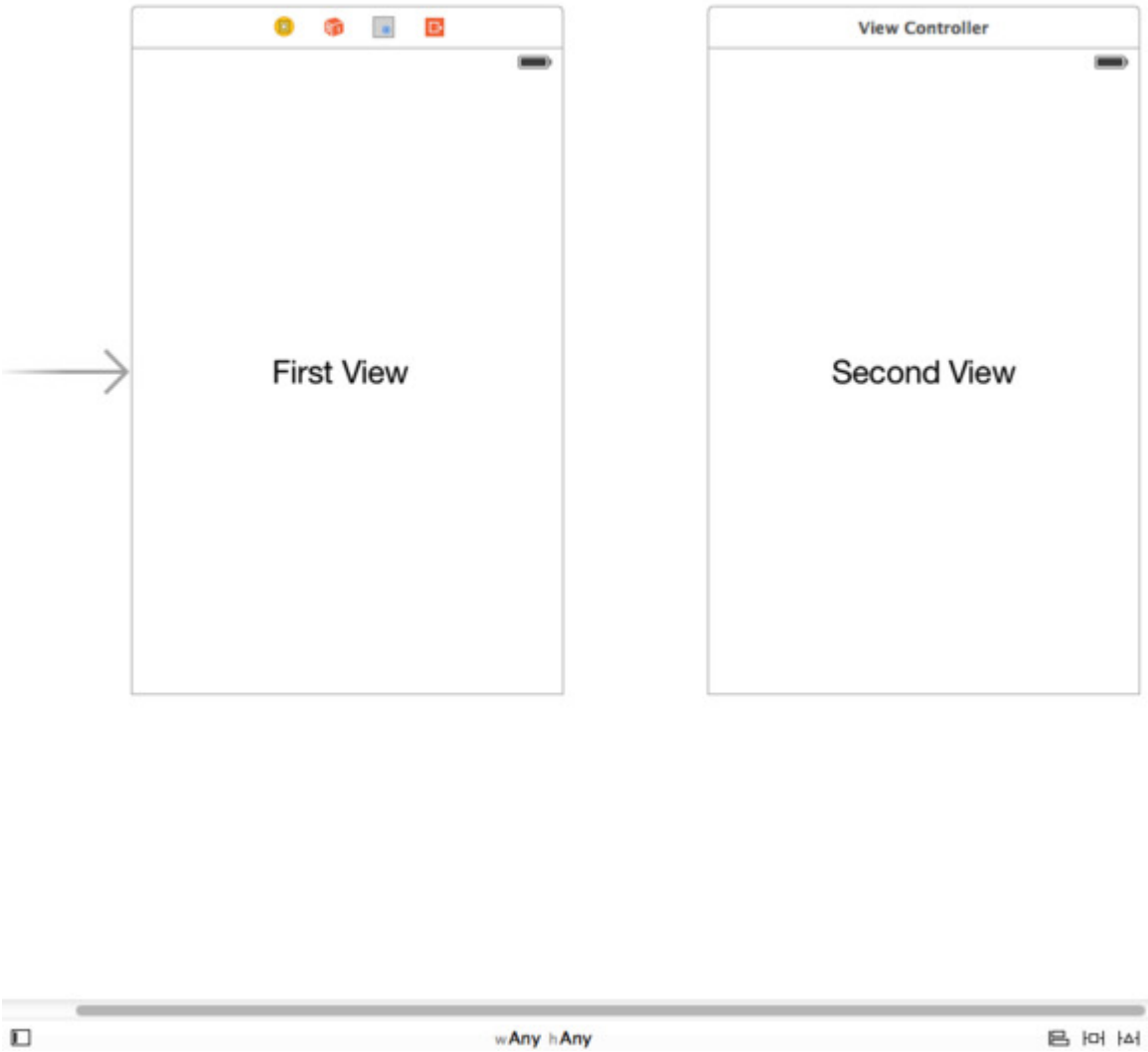


Рисунок 1.4.1 –Viewcontrollers

Создаем связь как обычно, ПКМ тянем стрелку от указанного элемента на рисунке 1.4.2.

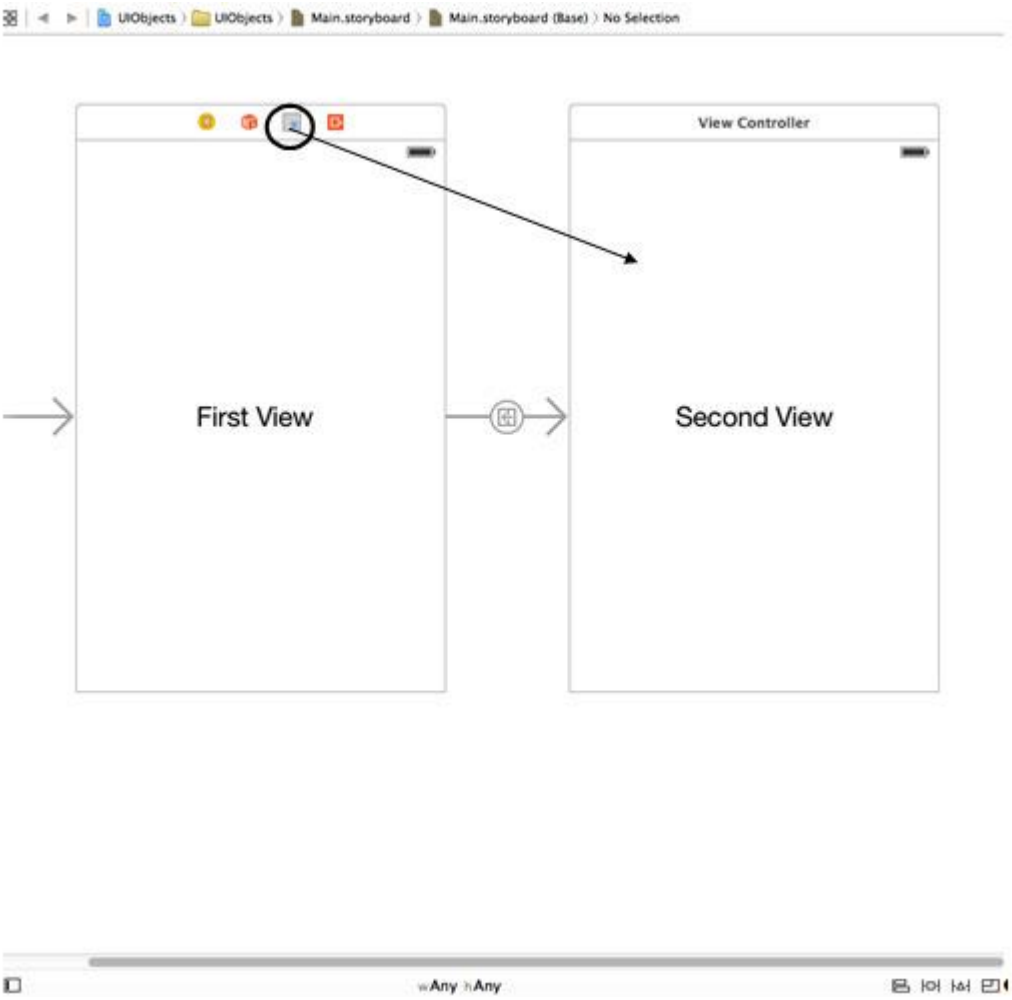


Рисунок 1.4.2 — Связь между Viewcontrollers

Выберете этот элемент на вью, в правом блоке выведутся атрибуты данного жеста. Основные атрибуты — это кол-во нажатий и сколько необходимо пальцев для осуществления нажатия. В соответствие с нормативным документом Apple — HumanInterfaceGuidelines, ваше приложение должно быть простым в использование, поэтому не рекомендуется выполнение каких-либо действий при помощи обильного количества GestureRecognizer, так же не стоит усложнять сами добавленные жесты. Если для наше тапа мы добавим кол-во необходимых касаний одновременно больше одного, то это приведет к неудобство в пользование приложения, и скорее всего ваше приложение будет отклонено.

Для нашего тапа выставите кол-во касаний равных двум в блоке атрибутов (рис. 1.4.3).

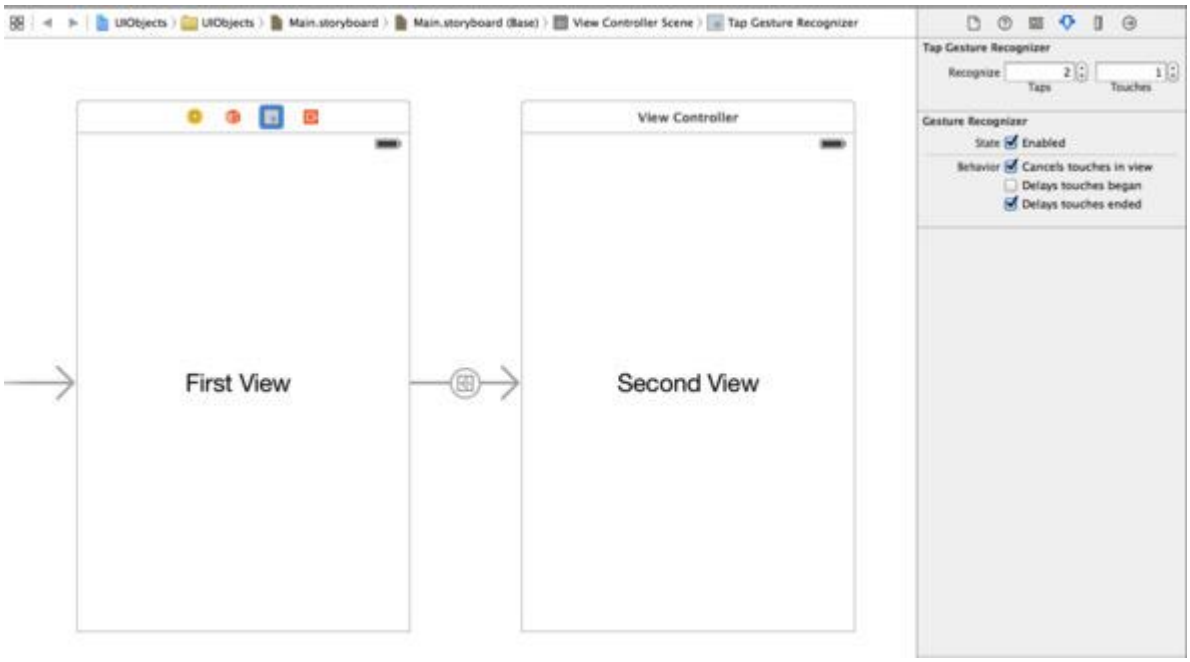


Рисунок 1.4.3 — Параметры жестов

Теперь на второй вью добавьте свайп и укажите для него переход на первый вью. Затем зайдите в атрибуты свайпа и укажите ему тип свайпа — снизу-вверх (рис. 1.4.4).

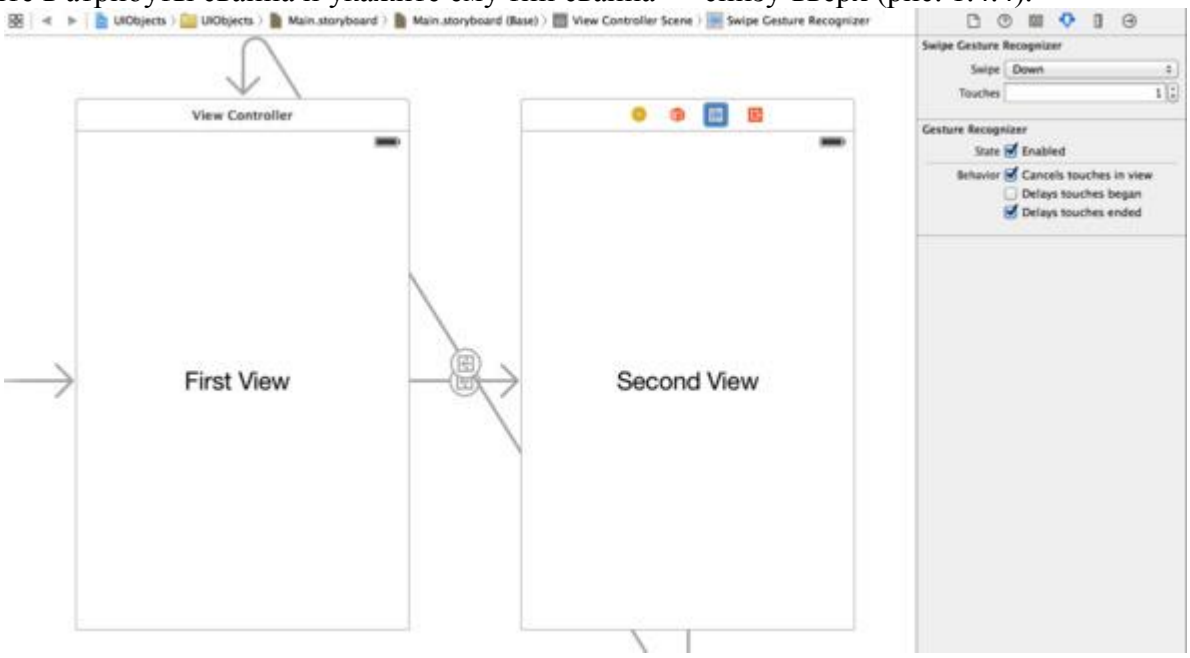


Рисунок 1.4.4 — Атрибуты свайпа

Теперь мы получили приложение, в котором мы можем переходить из одного вью, по двойному нажатию, во второй вью, и обратно, по проведению пальца снизу-вверх. Так же необходимо помнить, что приложение не стоит перегружать, оно должно быть простым в пользование.

Рассмотрим еще два типа жестов — это pinch (сужение/расширение) irotation (вращение).

Оставим в проекте один вью контроллер, удалим с него прошлые жесты. Добавьте на него UIImageView, желательно размер укажите в два раза меньше размера экрана = 160x240.Добавьте любую картинку такого же размера. Установите связь между ImageViewи ViewController.

Реализовывать мы будем через код, нам понадобиться создать объект класса UIPinchGestureRecognizer, инициализировать его для вью и задать селектор:

— (void) viewDidLoad {

[superviewDidLoad];
UIPinchGestureRecognizer *pinch = [[UIPinchGestureRecognizeralloc]
initWithTarget: self
action:@selector (pinch:)];
[self.viewaddGestureRecognizer: pinch];
}

При инициализации мы указываем с чем будет взаимодействовать объект, в данном случаи с самим вью контроллеров, затем указываем селектор, а после, добавляем его на представление.

Реализуем метод, которые будет вызываться, когда будет распознан жест pinch:

— (void) pinch: (UIPinchGestureRecognizer *) recognizer {
CGFloatnewScale = _mutableImage.image.scale + [recognizer scale] — lastScale;
CGFloatpositionX = _mutableImage.bounds. origin. x;
CGFloatpositionY = _mutableImage.bounds. origin. y;
CGFloat width = _mutableImage.bounds.size. width;
CGFloat height = _mutableImage.bounds.size. height;
_mutableImage.bounds = CGRectMake (positionX, positionY, width * newScale, height * newScale);
lastScale = [recognizerscale];
}

Первое, нам необходимо вычислить новый масштаб, чтобы плавно изменить его, должны знать предыдущие значение, дабы получить разность, на которую увеличим, либо уменьшим масштаб ImageView. Затем зададим новые «границы» для объекта. Мы не меняем позицию, поэтому присваиваем тоже значение, меняем только высоту и ширину, умножая их на новый масштаб, тем самым изменяется масштаб картинки на экране, записываем масштаб, получаем от recognizer. Заметим, что меняем масштаб UIImageView, а не самой картинки, но, этот класс устроен таким образом, что он подстраивается под картинку, добавленную в ImageView, так что изменяя масштаб ImageView, мы меняем масштаб и картинки.

Чтобы исполнять два касания сразу, необходимо зажать кнопку altи нажать ЛКМ.

Уберем pinch, удалим код, который реализует его. Добавимrotation:

— (void) viewDidLoad {
[superviewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
UIRotationGestureRecognizer *rotation = [[UIRotationGestureRecognizeralloc]
initWithTarget: self
action:@selector (rotation:)];
[self.viewaddGestureRecognizer: rotation];
}

Реализуемметод rotation:

— (void) rotation: (UIRotationGestureRecognizer *) recognizer {
CGAffineTransformtransformRotation = CGAffineTransformMakeRotation ([recognizer rotation]);
_mutableImage.transform = transformRotation;
}

СтруктураCGAffineTransformпозволяеттрансформироватьнашкартинку, имеетнесколькометодов, одинизнихCGAffineTranformMakeRotation, принимаетаргументврадианах. Далее мы присваиваем новую форму для ImageView, тем самым вращаем нашу картинку. Но будет некорректное вращение, чтобы оно было плавным, необходимо получать разницы между предыдущим и новым углом, и присваивать эту разницу.

AVAudioPlayer

Нативный аудиоплеер, позволяющий воспроизводить аудиофайлы в форматах mp3,wav, Вы можете заметить, что данный объект не относится к UIKitэлементам, а значит нам нужно будет подключить фреймворкAVFoundation. Он предоставляет возможности для работы с медиа-файлами: воспроизведение, редактирование.

Для создания нашего примера понадобится добавить на вью контроллер кнопку, нажав на которую, будет проигрываться музыка. Затем связать ее с вью. Далее добавитьфреймворк (рис. 1.5.1).

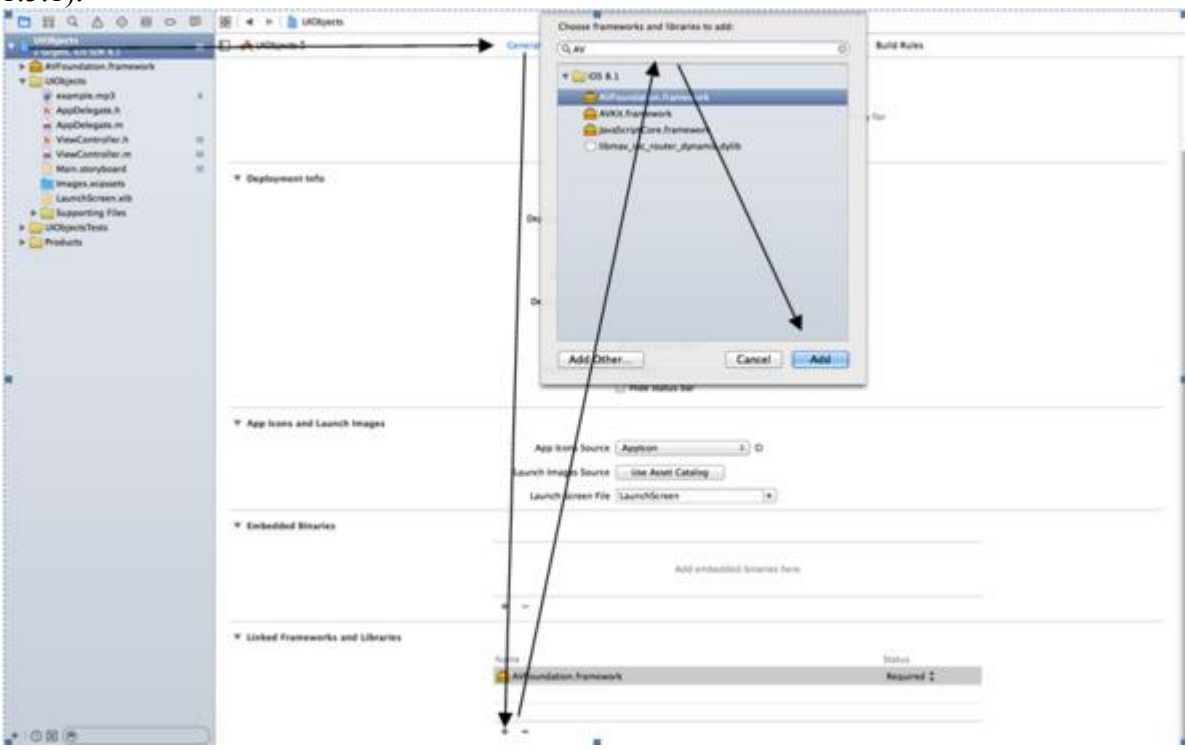


Рисунок 1.5.1 — Добавление фреймворка

Теперь подключите класс AVFoundation. Затем в метод viewDidLoad добавьте код:

— (void) viewDidLoad {
[superviewDidLoad];
NSBundle *mainBundle = [NSBundlemainBundle];
NSError *error;
NSString *fileName = @"example.mp3»;
NSURL *soundURL = [NSURLfileURLWithPath: [mainBundlepathForResource: fileNameofType: nil]];
player = [[AVAudioPlayeralloc] initWithContentsOfURL: soundURLerror:&error];
[playerprepareToPlay];
}

NSBundle — это объект, который предоставляет доступ к файловой системе устройства. Так как у AppleiOS– это консервативная система. То каждое приложение находится в своей «песочнице», то есть к другим файлам, записанные на телефоне, приложение имеет ограниченный доступ. Изменять файлы можно только в директории Documents вашего приложение, в других директориях файлы будут только считываться.

Мы создаем экземпляр класса NSBundle при помощи метода mainBundle, который указывает на расположение файлов, где находится само приложение.

NSError — предоставляет более подробную информацию об случающихся ошибках.

NSURL — указывает на удаленные, либо локальные данные.

Теперь создадим сам плеер, нужно учесть, что плеер обязан быть синглтоном (только одним единственным). Плеер инициализируется сразу с файлом. Так как аудиоплеер перед воспроизведением музыки, загружает аудиофайл в кэш, то это вызывает блокирование

приложения. Есть два способа избавиться от блокирования — это использовать второй поток, либо загружать аудиофайл во время загрузки экрана. Мы выбрали второй путь.

Осталось дело за малым, это запустить аудиофайл, добавьте следующий код в метод нашей кнопки:

```
— (IBAction) playMusic: (id) sender {
[playerplay];
}
```

Плеер можно запускать в определенное время, через метод playAtTime, который принимает аргумент в виде времени:

```
[playerplayAtTime:player.deviceCurrentTime +1];
```

Через одну секунду после старта, будет запущен аудиофайл.

Рассмотрим еще пример про громкость.

Добавим на наш вью UISlider. Этот объект принадлежит UIKit, представляет собой регулятор значения. Установить связь слайдера с вью как outlet, так и action. Action будет вызываться каждый раз, как только будет изменено положение регулятора на слайдере.

В методе слайдера и кнопки пропишите:

```
— (IBAction) changeVolume: (id) sender {
player. volume = _sliderVolume.value;
}
— (IBAction) playMusic: (id) sender {
player. volume = _sliderVolume.value;
[playerplay];
}
```

Теперь мы можем регулировать громкость плеера с помощью слайдера.

UICollectionView

Класс UICollectionViewпредставляет упорядоченный набор данных, делает это он с помощью настраиваемых макетов. UITableViewпредставляет собой частный случай UICollectionView. Примером такого класса служит галерея в устройствах Apple.

Создаете новый проект, затем удалите из него текущий контроллер. Добавьте CollectionViewController. Создайте новый класс, который является наследником «UICollectionViewController», и присвойте его текущим контроллеру (рис. 1.6.1).

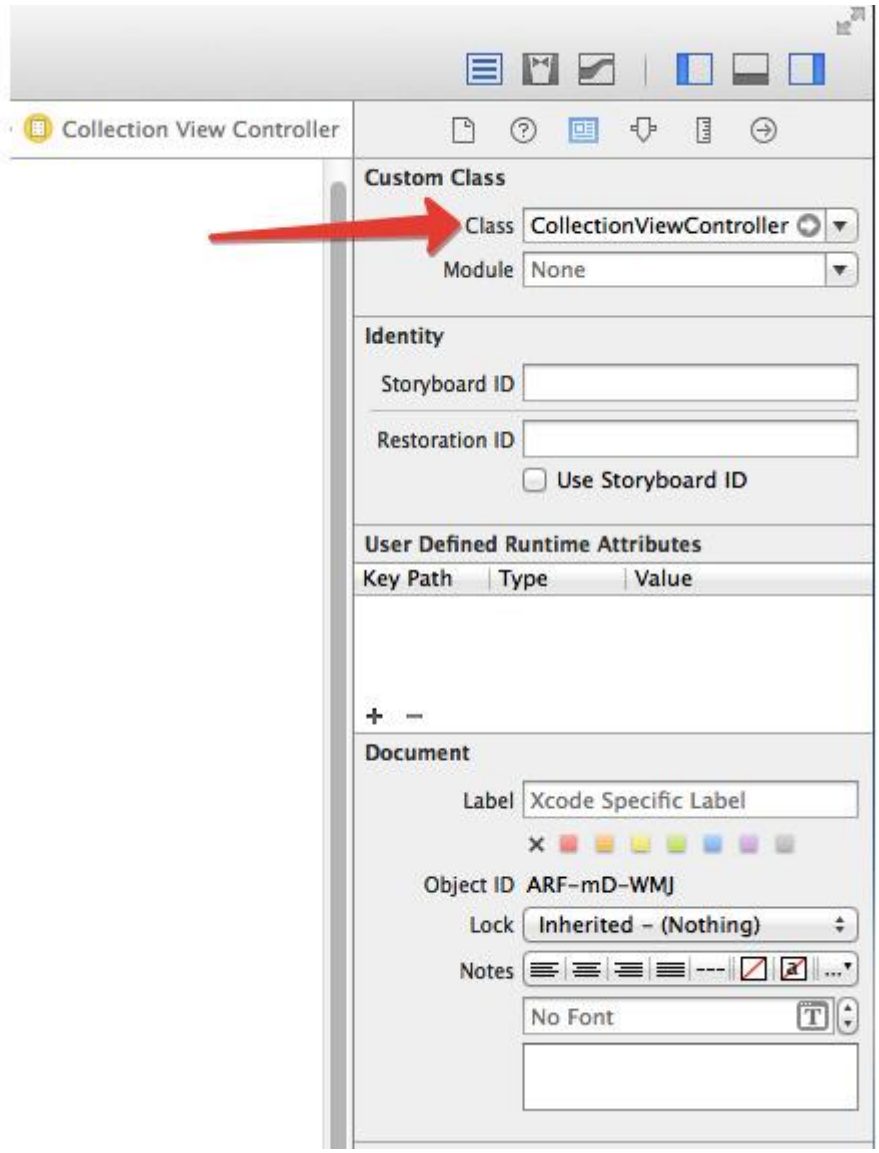


Рисунок 1.6.1 — Присвоения класса.

На самом контроллере есть ячейка — «UICollectionViewCell». Задайте ей идентификатор — «Cell». У такой ячейки нет таких параметров как: image, text. Поэтому, чтобы вывести изображение, добавьте «UIImageView» на ячейку. Задайте ему тег 1.

Теперь выведем изображение на экран. В методе «viewDidLoad», добавьте код:

— (void) viewDidLoad {

[superviewDidLoad];

images = [NSArrayarrayWithObjects:@"image1.png",@"image2.png",@"image3.png",@"imgae4.png",@"image5.png», nil];

}

Как видите, создали массив, в котором хранятся названия изображений. Так можете заметить, что нет такой строчки кода:

[self.collectionViewregisterClass: [UICollectionViewCellclass]

forCellWithReuseIdentifier: reuseIdentifier];

Этот метод задает ячейке класс «UICollectionViewCell», потому создается новая ячейка, в которой нет «UIImageView».

Как и у таблицы, есть два метода, которые отвечают за кол-во строк и секций. Первый метод обязательный, второй необязательный. Количество строк пусть будет равно количеству картинок. Количество секций равно единице. И есть еще один метод

«cellForItemAtIndexPath». В нем будет заполняться вью. Добавьте в этот метод следующий код:

— (UICollectionViewCell *) collectionView: (UICollectionView *) collectionViewcellForItemAtIndexPath: (NSIndexPath *) indexPath {

UICollectionViewCell *cell = [collectionViewdequeueReusableCellWithReuseIdentifier: reuseIdentifierforIndexPath: indexPath];

UIImageView *recipeImageView = (UIImageView *) [cell viewWithTag:100];// получаемобъектпотегу.

recipeImageView.image = [UIImageimageNamed:[imagesobjectAtIndex:indexPath.row]];// задаемкартинку

returncell;

}

Теперь у нас есть элементарная фотогалерея. Далее сделаем редактируемую фотогалерею и детальный просмотр. Начнем с последнего.

Добавьте viewcontroller с imageview, добавьте еще свойство для «UIImage». В методе «viewDidLoad», укажите, чтобы «UIImage» присваивалось "imageView.image». Следующий шаг — это добавить navigationcontrollerдля collectionview. Чтобы могли через код переходить к viewcontroller. Реализуйте код:

— (void) collectionView: (UICollectionView *) collectionViewdidSelectItemAtIndexPath: (NSIndexPath *) indexPath {

if (!isEditable) {// заранее указываем проверку, редактируются ли сейчас данные.

DetailViewController *detailView = [[UIStoryboardstoryboardWithName:@«Main"bundle: nil] instantiateViewControllerWithIdentifier:@«detailViewController»]; // указываемсториборд, второйпараметрпринимаетпуткфайлу, передаемнил (будетвыбраносновнойсториборд). Получаемконкретныйконтроллерпоиндентификатору.

detailView.selectedImage = [UIImageimageNamed: [imagesobjectAtIn dex:indexPath.row]]; // передаемкартинку

[self.collectionViewdeselectItemAtIndexPath: indexPathanimated: NO]; // деселект

[self.navigationControllerpushViewController: detailViewanimated: YES]; // переходимспомощьюнавигации.

}

Реализуем редактируемыйcollectionview. Понадобиться добавить кнопку на навигационную панель, добавить для нее действие. Необходимо добавить флаг «isEditable». При нажатие на кнопку, должны устанавливать флаг в определенное состояние, добавлять кнопку «delete»:

— (IBAction) edit: (UIBarButtonItem *) sender {

if (isEditable) {

isEditable = NO;

self.navigationItem. rightBarButtonItem = nil; // убираемкнопку

self.collectionView.allowsMultipleSelection = NO;

[self.collectionViewreloadData];

} else {

isEditable = YES;

UIBarButtonItem *delete = [[UIBarButtonItemalloc] initWithTitle:@«Delete»

style:0// так как представляет собой список из интов

target: self// объект, принимающий сообщение

action:@selector (deleteFoto)]; // метод указываем

self.navigationItem. rightBarButtonItem = delete; // добавляем кнопку на навигационную панель

self.collectionView.allowsMultipleSelection = YES; // множественныйвыбор

[self.collectionViewreloadData]; // перезагружаемданные

}

}

Осталось реализовать метод удаления изображений. Добавьте такой код:

— (void) deleteFoto {

NSMutableArray *mutableImages = [NSMutableArrayarrayWithArray: images]; // создаем изменяемый массив

if (selectedIndexPaths.count> 0) {// проверяем, выбранны ли какие-нибудь картинки

for (NSIndexPath *indexPathinselectedIndexPaths) {// перебираемвыбранныйпозиции

[mutableImagesremoveObjectAtIndex:indexPath.row];

}

images = [NSArrayarrayWithArray: mutableImages]; // создаемновыймассив

[selectedIndexPathsremoveAllObjects];

[self.collectionViewreloadData];

}

}

— (void) collectionView: (UICollectionView *) collectionViewdidSelectItemAtIndexPath: (NSIndexPath *) indexPath {

if (!isEditable) {

DetailViewController *detailView = [[UIStoryboardstoryboardWithName:@«Main"bundle: nil] instantiateViewControllerWithIdentifier:@«detailViewController»]; // указываемсториборд, второйпараметрпринимаетпутькфайлу, передаемнил (будетвыбраносновнойсториборд). Получаемконкретныйконтроллерпоиндентификатору.

detailView.selectedImage = [UIImageimageNamed:[imagesobjectAtIndex:indexPath.row]]; // передаемкартинку

[self.collectionViewdeselectItemAtIndexPath: indexPathanimated: NO]; // деселект

[self.navigationControllerpushViewController: detailViewanimated: YES]; // переходимспомощьюнавигации.

} else {

[selectedIndexPathsaddObject: indexPath];

}

}

— (void) collectionView: (UICollectionView *) collectionViewdidDeselectItemAtIndexPath: (NSIndexPath *) indexPath {

if (isEditable) {

[selectedIndexPathsremoveObject: indexPath];

}

}

Впервомметодеудаляемвыбранныефотографии. Во втором методе происходит выбор, а в третьем обратное.

Таким образом, у нас есть галерея, в которой можно удалять фотографии, и детально просматривать их.

UIWebView

Класс, предоставляющий набор функция для просмотра данных. Данными могут являться локальными, и из интернета. Класс может читать различные форматы документов: pdf, doc, xls, rft, и другие. Есть возможность запуска java-script. Принимает на себя роль браузера.

Начнем с просмотра страниц в интернете. Добавьте web view на controller, подключите к view controller navigation controller, и укажите в атрибутах, чтобы был toolbar. Нам он пригодится для навигации по интернету.

Во view controller с webview, добавьте на toolbar четыре кнопки «UIBarButtonItem»: назад, отмена, обновить, вперед. И каждой задайте определенный identifier, подходящий

по стилю для навигации браузера, и тег. Далее свяжите все элементы с viewcontroller. Для кнопок создайте массив связей, и действие.

Теперь, откроем любую страницу из интернета:

— (void) viewDidLoad {

[super viewDidLoad];

// Do any additional setup after loading the view, typically from a nib.

self.internet.delegate = self;

// [self updateButtons];

NSURL *url = [NSURL URLWithString:@"https://developer.apple.com/"]; // Объект, содержащий ссылку на внешний ресурся, либо на локальный (файлы на телефоне)

NSURLRequest *request = [NSURLRequest requestWithURL: url]; // Представляет собой запрос, независимо от протокола и ссылки.

[self.internet loadRequest: request]; // метод, выполняющий запрос асинхронно.

}

URL может указываться как абсолютный, так и относительный (без протокола, корень сайта). При инициализации, необходимо указывать абсолютную ссылку.

У web view есть три метода загрузки данных. Первый — это загрузка через ссылки, второй — это загрузка html кода через строку. Третий — это загрузка через «NSData»:

— (void) loadData: (NSData *) data MIMEType: (NSString *) MIMEType textEncodingName: (NSString *) textEncodingName baseURL: (NSURL *) baseURL {

// принимает данные, представленные в бинарном виде. Указываем тип данных (например: для pdf — «application/pdf»). Указываем кодировку. Передаем основной юрл (если такой есть).

}

Останавливаться на них не будем. Займемся навигацией. Необходимо подключить протокол «UIWebViewDelegate», для реализации трех методов:

— (void) webViewDidStartLoad: (UIWebView *) webView {

[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible: YES]; // показать индикатор загрузки, когда начинается загрузка

[self updateButtons];

}

— (void) webViewDidFinishLoad: (UIWebView *) webView {

[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible: NO];

[self updateButtons];

[self.internet stringByEvaluatingJavaScriptFromString:@«alert («Finish load’);»]; // исполняет java-script код

}

— (void) webView: (UIWebView *) webView didFailLoadWithError: (NSError *) error {

[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible: NO];

[self updateButtons];

}

Как можете заметить, у web view есть метод, исполняющий java-script код, который храниться в строке. Еще метод «updateButtons», необходимо реализовать его, и действие, происходящее по нажатию на одну из кнопок:

— (IBAction) actionBrowser: (UIBarButtonItem *) sender {

switch (sender. tag) {

case UIBarButtonItemContentStyleReward:

[self.internet goBack];

break;

case UIBarButtonItemContentStyleCansel:

[self.internet stopLoading];

break;

```
case UIBarButtonItemContentStyleRefresh:
[self.internet reload];
break;
case UIBarButtonItemContentStyleFastReward:
[self.internet goForward];
break;
default:
break;
}
[self updateButtons];
}
— (void) updateButtons {
for (UIBarButtonItem *sender in self.browserButtons) {
switch (sender. tag) {
case UIBarButtonItemContentStyleReward:
sender. enabled = self.internet.canGoBack;
break;
case UIBarButtonItemContentStyleFastReward:
sender. enabled = self.internet.canGoForward;
break;
case UIBarButtonItemContentStyleCansel:
sender. enabled = self.internet. loading? YES: NO;
break;
default:
break;
}
}
}
```

Switch-конструкция реализована на теге кнопок. Для лучшей читабельности кода, добавьте список в сопоставление с действиями кнопок.

В первом методе реализуется действия, такие как: назад, отмена, перезагрузка, вперед. Во втором методе происходит обновление состояния кнопок. Итог программы на рисунке 1.7.1.

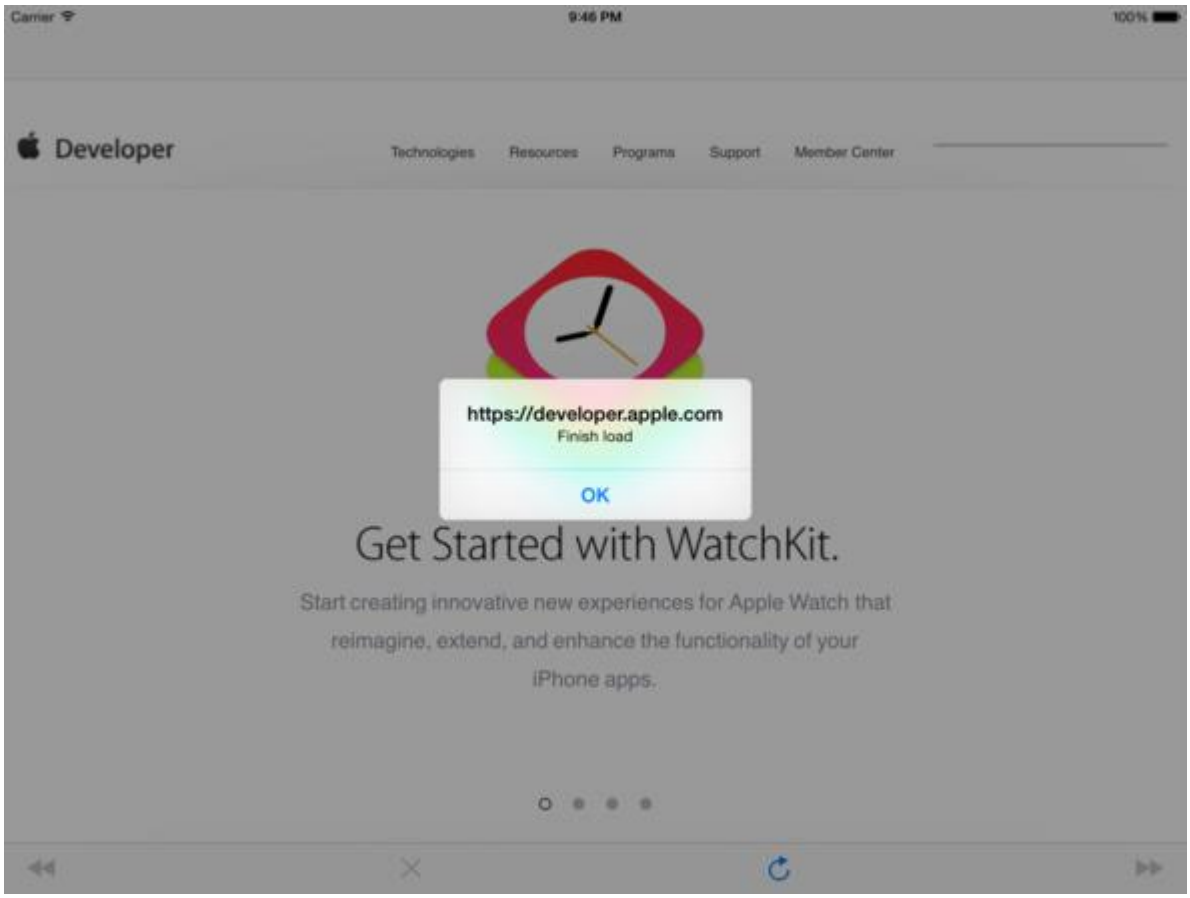


Рисунок 1.7.1 — Результат программы

Есть еще один важный метод:

— (BOOL) webView: (UIWebView *) webView shouldStartLoadWithRequest: (NSURLRequest *) request navigationType: (UIWebViewNavigationType) navigationType {

// метод, который вызывается перед началом выполнения запроса. navigationType — возвращает тип действия, совершенный пользователем для нового запроса

return YES;

}

На этом закончим рассматривать «UIWebView» и перейдем базе данных «Core Data».

Классы. Методы. Свойства

Один из важных понятий в ООП — это класс. Класс — это разновидность абстрактных типов данных. Суть отличия классов от других абстрактных типов данных состоит в том, что при задании типа данных класс определяет одновременно и интерфейс, для этого создается два файла с расширениями: h (заголовочный файл, определяющий интерфейс класса) и m (исходный файл, определяющий реализацию класса). И реализацию для всех своих экземпляров.

Классы

Интерфейс класса — семантическая и синтаксическая конструкция в коде программы, используемая для специфицирования услуг, предоставляемых классом или компонентом. Другим словами, интерфейс класса предоставляет набор доступных методов и свойств этого класса.

Реализация — код программы, реализующие методы, объявленные в интерфейсе класса, и реализующие инкапсулированные методы, которые не объявлены в интерфейсе класса.

Разберем на примере. Создадим пустой проект, или откроем уже имеющийся и выберем файл с расширением AppDelegate. h (рис. 1).

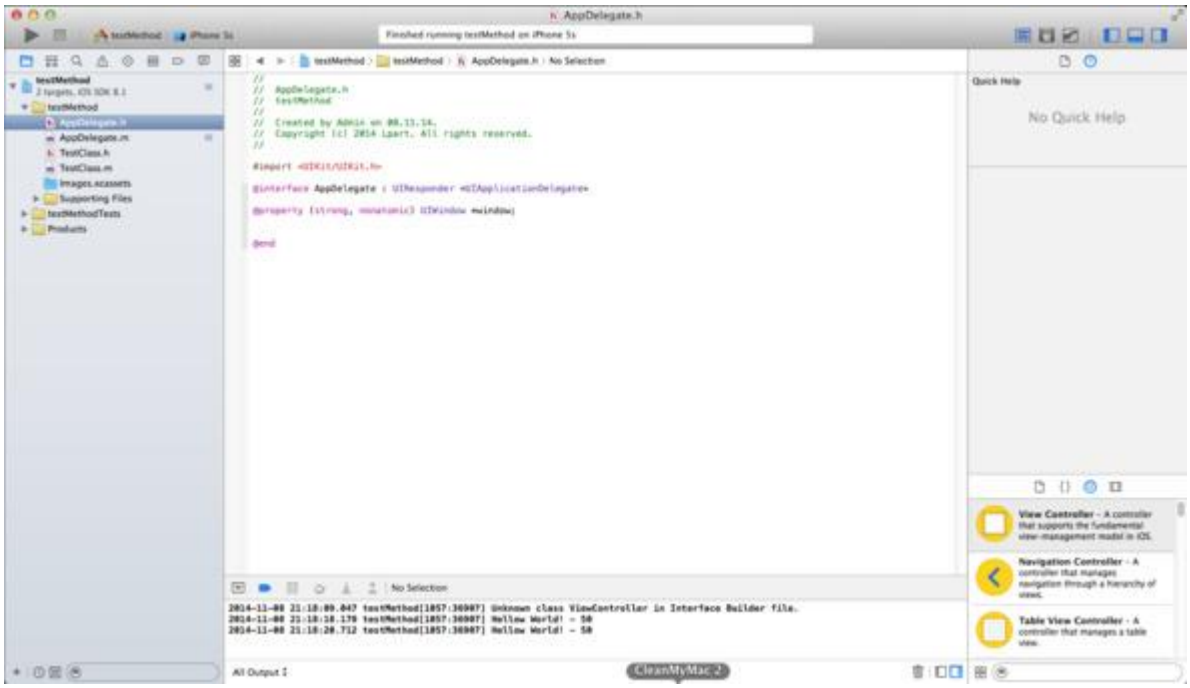


Рисунок 1 — Интерфейс класса AppDelegate

Закомментированная шапка — это параметры вашего проекта.

Директива import, более совершенна чем include, она исключает рекурсивное подключение классов.

@interface AppDelegate: UIResponder <UIApplicationDelegate>

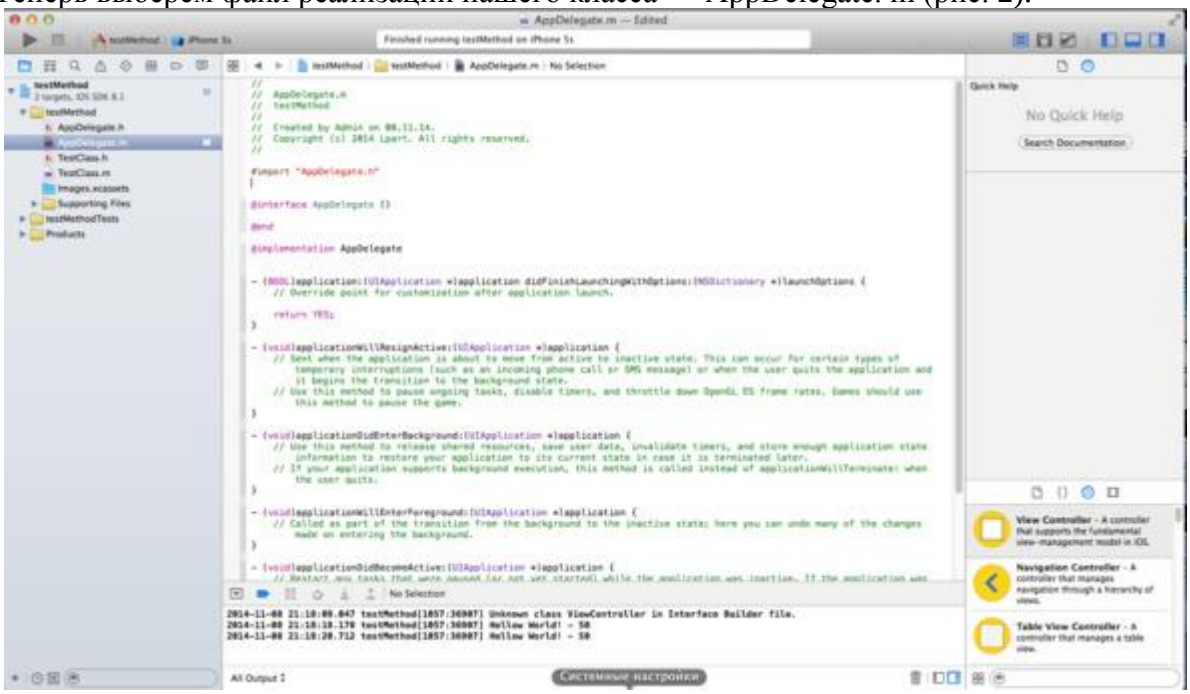
Ключевое слово интерфейс — служит определением класса. AppDelegate — название класса, после двоеточия идет названия суперкласса или название наследуемого класса, к которому вернемся чуть позже. А в <> указывается название протоколов, разбираться с ними будем позже.

@property (strong, nonatomic) UIWindow *window;

Далее идет свойство с определенными параметрами, к ним вернемся позже. В данном свойстве, указывается имя экземпляра класса или объекта и само название класса.

После идет ключевое слово, которое указывает на окончание блока определения класса — @end.

Теперь выберем файл реализации нашего класса — AppDelegate. m (рис. 2).



Р

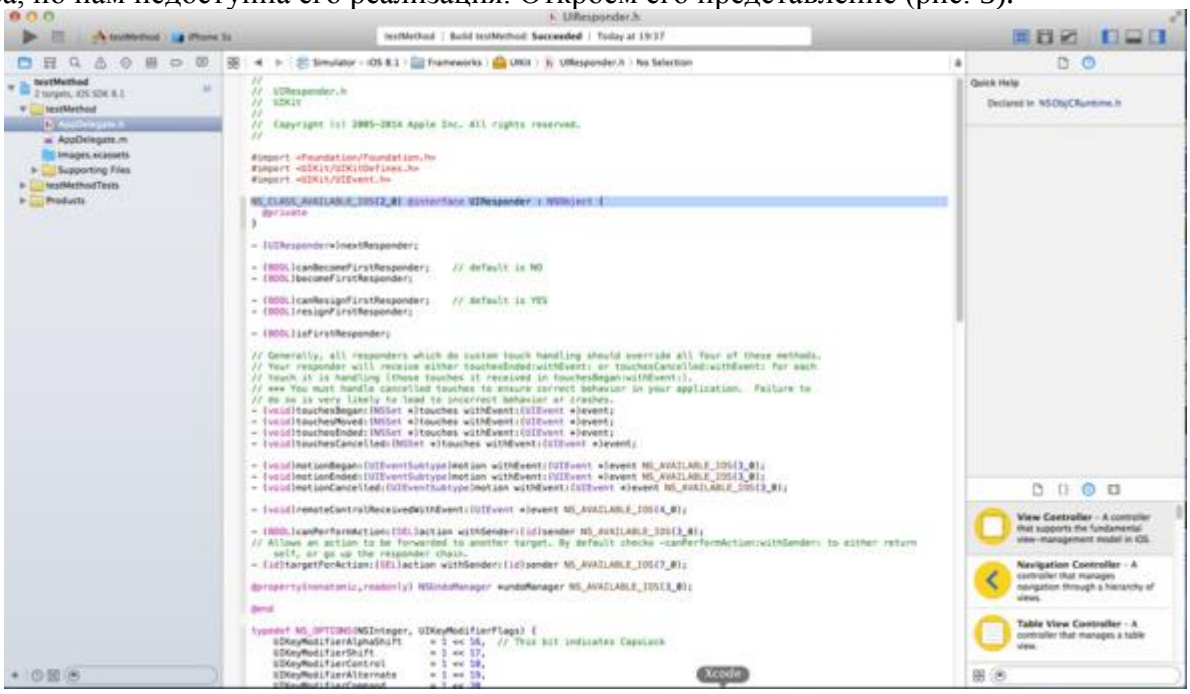
исунок 2 — Реализация класса AppDelegate

Знакомая нам уже шапка, как и директива import, которая добавляет представление класса. Затем снова идет интерфейс класс, только здесь уже нужен он для расширения класса.

@implementation AppDelegate

Ключевое слово реализация, затем название класса.

Вернемся к нашему суперклассу UIResponder. Мы можем посмотреть интерфейс этого класса, но нам недоступна его реализация. Откроем его представление (рис. 3).



Р

исунок 3 — Интерфейс класса UIResponder

Вы можете увидеть, что представление содержит в себе методы и свойства, которые доступные в классе AppDelegate. Он наследует все методы и классы, объявленные в представлении класса UIResponder. Таким образом, получается, что мы можем не объявлять методы в представление AppDelegate, чтобы иметь к ним доступ, так и со свойствами.

В нашем случае получается, что используется один из трех главных, основополагающих принципов ООП — наследование.

Создадим свой класс и назовем его TestClass (рис..4), для этого нажмем File-> New file-> iOS-> Source-> Cocoa Touch Class-> Next, язык выбираем objective-c, наш класс будет наследником NSObject (является основным классом Objective-C).

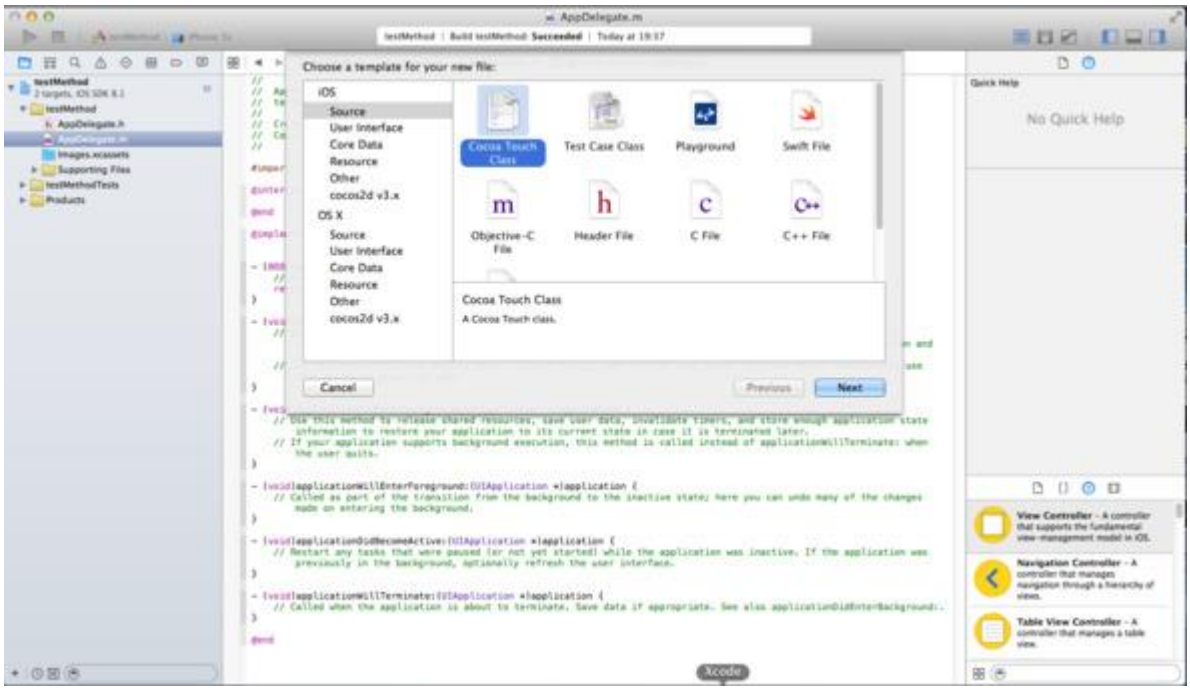


Рисунок 4 — Создание файлов в Xcode

Методы

Создадим наш первый метод, для этого в TestClass. h (рис 5) нам нужно объявить метод, а в TestClass. m (рис 6) реализовать его.

Рисунок 5 — Объявление методов в интерфейсе класса

Как видите, мы объявили два не инкапсулированных: метода

— (void) firstMethod;

— (void) secondMethodWithString: (NSString *) string number: (NSInteger) magicNum;

Разберем сразу, что они себя представляют:

1) "-" означает, что метод принадлежит экземпляру этого класса, если бы было "+», то метод принадлежал бы самому классу, чуть позже, рассмотрим на примере.

2)» (void)» в скобках указывается, что будет возвращать нам метод, в данном случаи он ничего не будет возвращать.

3) Далее следует само название метода.

4) Во втором нашем методе, мы уже передаем два аргументы, один **объект** NSString с именем string, поэтому стоит **указатель**, и один примитивный тип данных NSInteger с именем magicNum. Number является продолжением названия метода.

Теперь реализуем их (рис 6)

Рисунок 6 — Реализация методов

Код программы:

— (void) firstMethod {

[self secondMethodWithString:@«Hello World!» number:50];

}

— (void) secondMethodWithString: (NSString *) string number: (NSInteger) magicNum {

NSLog (@"%@ — %ld», string, (long) magicNum);

}

Разбираемся, наш первый метод вызывает второй метод, передавая ему строку «Привет Мир» и число равное 50.

[self secondMethodWithString:@«Hello World!» number:50];

Чтобы вызвать метод объекта, мы используем self. Self — указатель на сам объект этого класса, затем передаем сообщение о вызове метода объекту и передаем параметры.

Еще используются такие конструкции:

[self performSelector: @selector (secondMethodWithString:) withObject:@«Hello World!»];

В этом случае, мы опять же обращаемся к объекту, но теперь уже посылаем сообщение о вызове метода через селектор. Мы даем знать компилятору, что secondMethodWithString является селектором, такая конструкция, позволяет компилятору быстрее найти метод, зная, что он является селектором. Далее мы передаем нашу строку.

Во втором методе используем команду NSLog, которая распечатывает нам строку в консоли (рис. 7) с принятыми аргументами. В самой распечатываемой строке, мы указываем форматы строки через процент, первый формат "%@" принимает значение строк, а второй формат "%ld» принимает число типа long. Через запятую указываем передаваем параметры, (long) приводит наше magicNum к типу long.

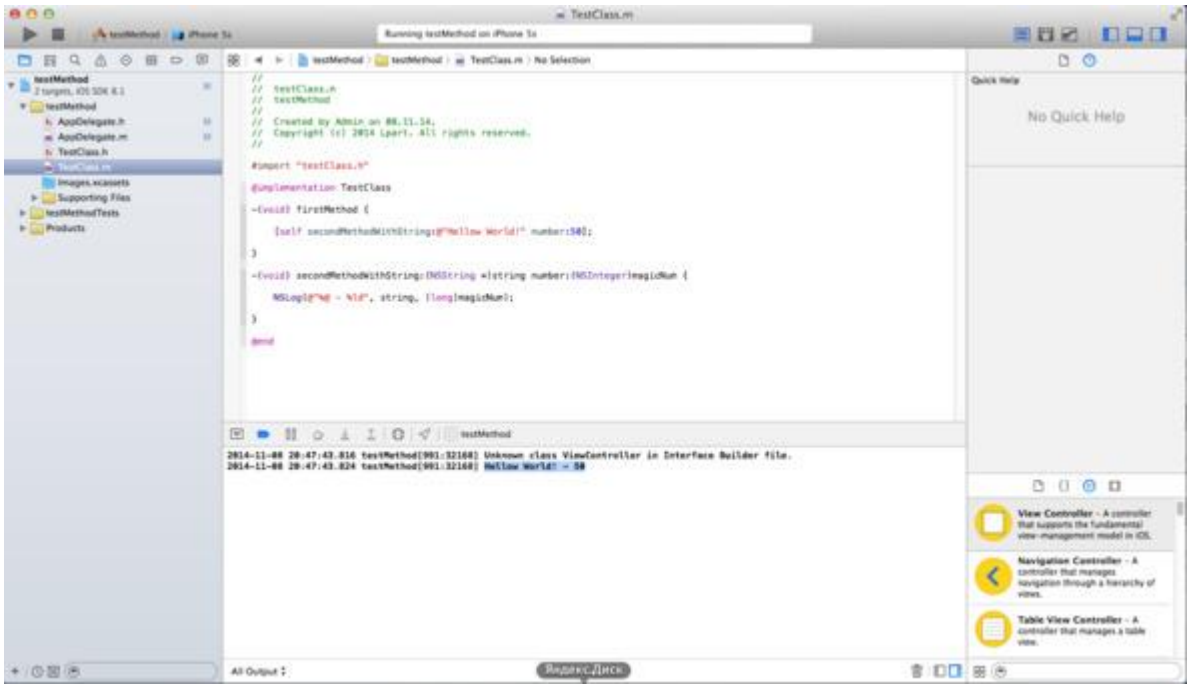


Рисунок 7 — Вывод в консоль

Теперь вернемся в AppDelegate. m и импортируем наш свежеиспеченный класс. Импортировать надо TestClass. h, таким образом мы не будем засорять пространство имен, так как единственное, что импортировано TestClass. h — Foundation, нежели TestClass. m, в который может содержать большое кол-во других классов.

Создадим экземпляр класса в методе, как на рисунке (рис. 8):

— (BOOL) application: (UIApplication *) application didFinishLaunchingWithOptions: (NSDictionary *) launchOptions

Этот метод вызывается всегда, когда запускается приложение.

Продолжим, для этого мы напишем такой код:

TestClass *object = [[TestClass alloc] init];

Теперь у нас есть экземпляр класса TestClass

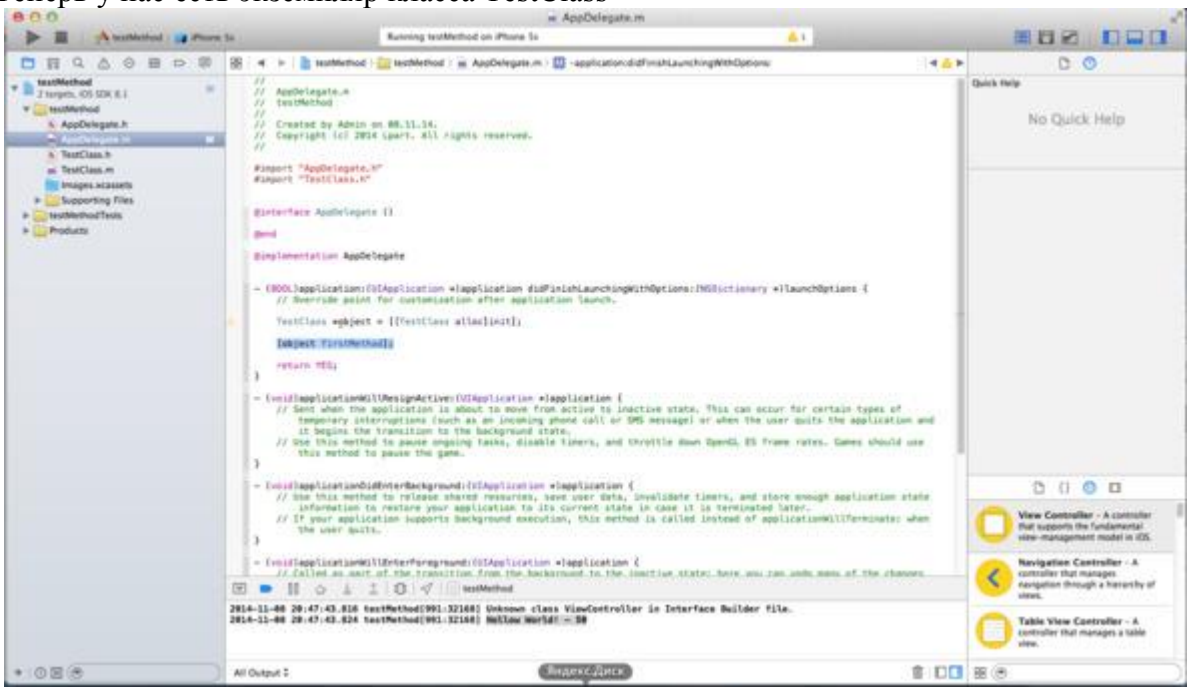


Рисунок 8 — Экземпляр класса

В нашей строчке, мы указываем название класса, затем ставим указатель «*», после имя, а дальше мы обращаемся к нашему классу и вызываем команду alloc, которая отвечает

за выделение памяти объекту, а дальше обращаемся к уже созданному объекту с командой инициализации.

Почему мы можем вызвать две эти команды, вспомните один из принципов ООП — наследование. Наш класс наследовал методы класса NSObject и теперь тоже их содержит.

Выполним еще один шаг — переопределение метода (рис. 9). Переопределять мы будем метод init в нашем классе. Для этого вернемся в наш класс, в его реализацию, объявлять метод необязательно, т.к. он уже объявлен в представлении NSObject. Напишем такой код:

— (id) init {
1) self = [super init];
2) if (self) {
[self firstMethod];
}
3) return self;
}

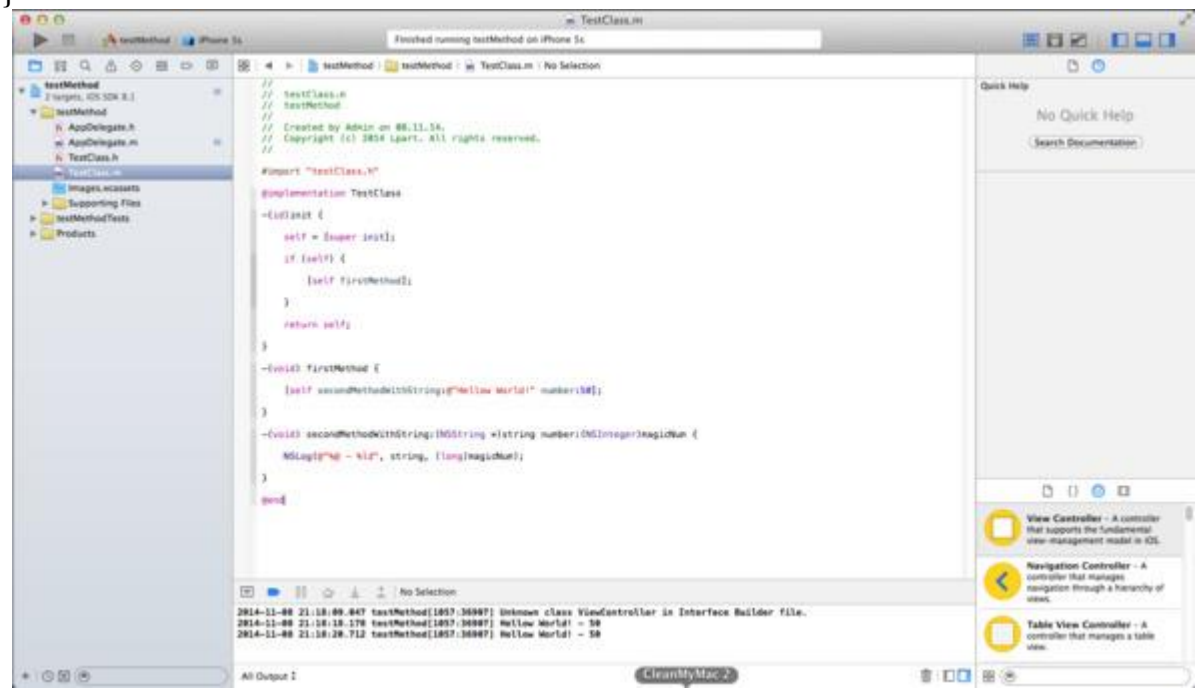


Рисунок 9 — Переопределение метода

Разберем метод init. Он возвращает нам id. Id — указатель на любой объект. Далее реализуем переопределенный метод:

1) Еще одно ключевое слово — super. Это указатель на родительский класс, или суперкласс, точнее объект этого класс. Мы обращаемся к его команде init, инициализируем наш объект. Если не будет выполнен этот метод у родителя, то объект не будет инициализирован.

2) Если объект инициализирован, то выполнить вызов метода нашего класса.

3) Вернем наш объект, в том случаи, если он не будет инициализирован, то вернется указатель на адрес память, значение которого равно 0

До переопределения был один метод init в классе NSObject, после их стало два. Как компилятор определяет какой из них вызвать? А делает он это следующим образом:

Он ищет этот метод в нашем классе testClass, и, если он находит, то исполняет его, иначе, если не нашел, он переходит в родительский класс в NSObject и ищет там метод init.

Теперь у нас в консоль дважды выводится строка с двумя аргументами.

Теперь продемонстрируем пример с методами самого класса. Для этого объявим и реализуем наш метод, который возвращаем нам строку, в виде такой конструкции, в классе testClass:

+ (NSString *) classMethod {
return @«I’m method of class, eeee!»;

}

Не забудьте объявить в представление класса!

Теперь мы можем вызвать этот метод в классе AppDelegate без создания экземпляра класс, обращаясь к TestClass (рис 10):

NSString *string = [TestClass classMethod];

NSLog (@"%@», string);

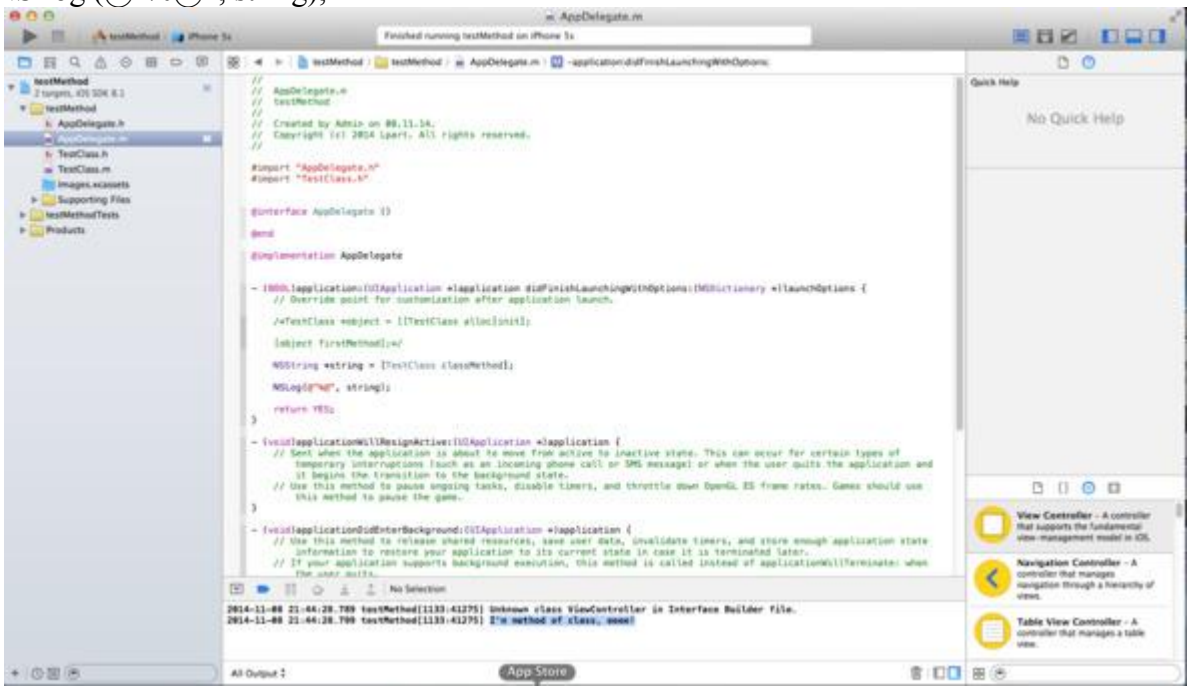


Рисунок 10 — Вызов метода класса

Свойства

Вернемся к нашим свойствам. Свойство, грубо говоря это публичные переменные класса, с определенно задаваемыми параметрами, имеющие setter/getter, которые мы можем переопределить.

Создадим три property, первому свойству переопределим геттер, а третьему зададим новый геттер, и инициализируем в методе init (рис 11, 12):

@property (strong, nonatomic, getter=getStrongString) NSString *strongString;

@property (weak, nonatomic) NSString *weakString;

@property (assign, nonatomic, setter=setNewNum:) NSInteger assignNum;

Рисунок 11 — Инициализация свойств

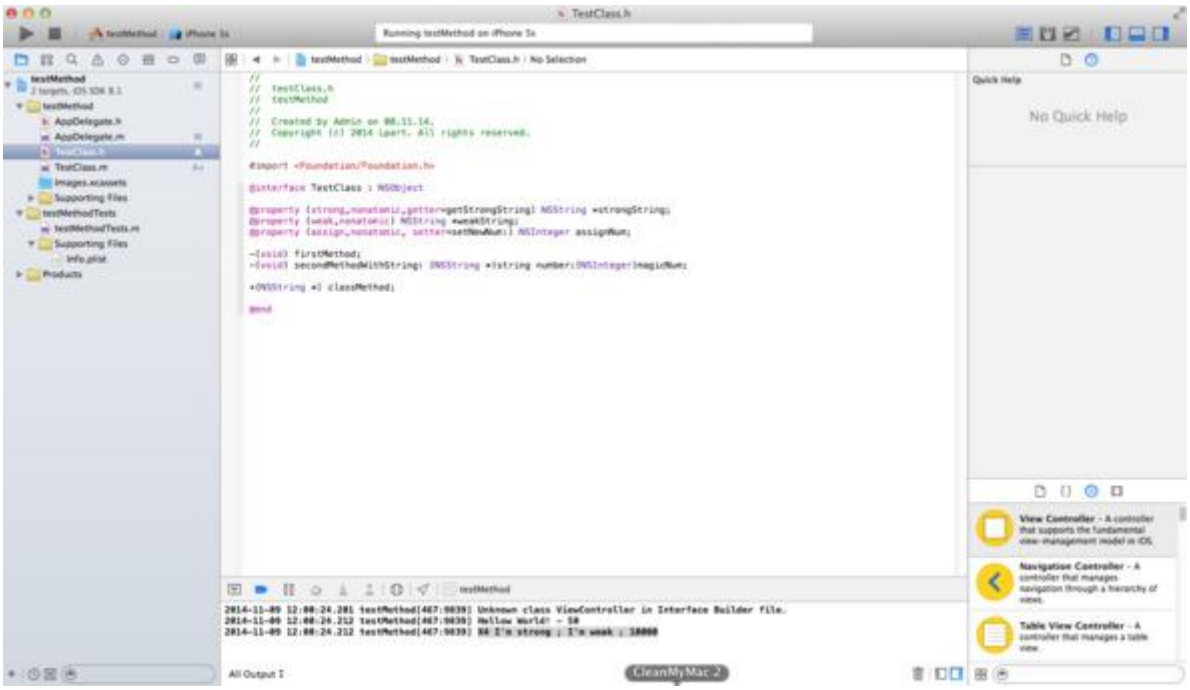


Рисунок 12 — Создание свойств

Рассмотрим свойство _assignNum. Нижнее подчеркивание означает, что мы обращаемся к свойству напрямую, вместо того, чтобы обращаться к объекту, к его свойству. Обратим внимание на сеттер для этого свойства, как видите, сеттер принимает аргумент, который присваивает нашему свойству. Если бы мы написали такой код:

```
— (void) setNewNum: (NSInteger) assignNum {
//_assignNum = assignNum; правильный код
self.assignNum = assignNum; // неправильный код
}
```

То в этом случае, мы обращались бы к свойству объекта вызывая его сеттер, в котором мы обращаемся к свойству и т.д., создав тем самым рекурсию. Этот сеттер будет вызываться до тех пор, пока есть свободная память у устройства, после программа упадет.

Вызовем сеттеры и геттеры наших свойств у объекта в классе AppDelegate:

```
NSString *strongString = [object getStrongString];
NSString *weakString = [object weakString];
[object setNewNum:10000];
```

Теперь выведем в консоль наши свойства (рис 13)

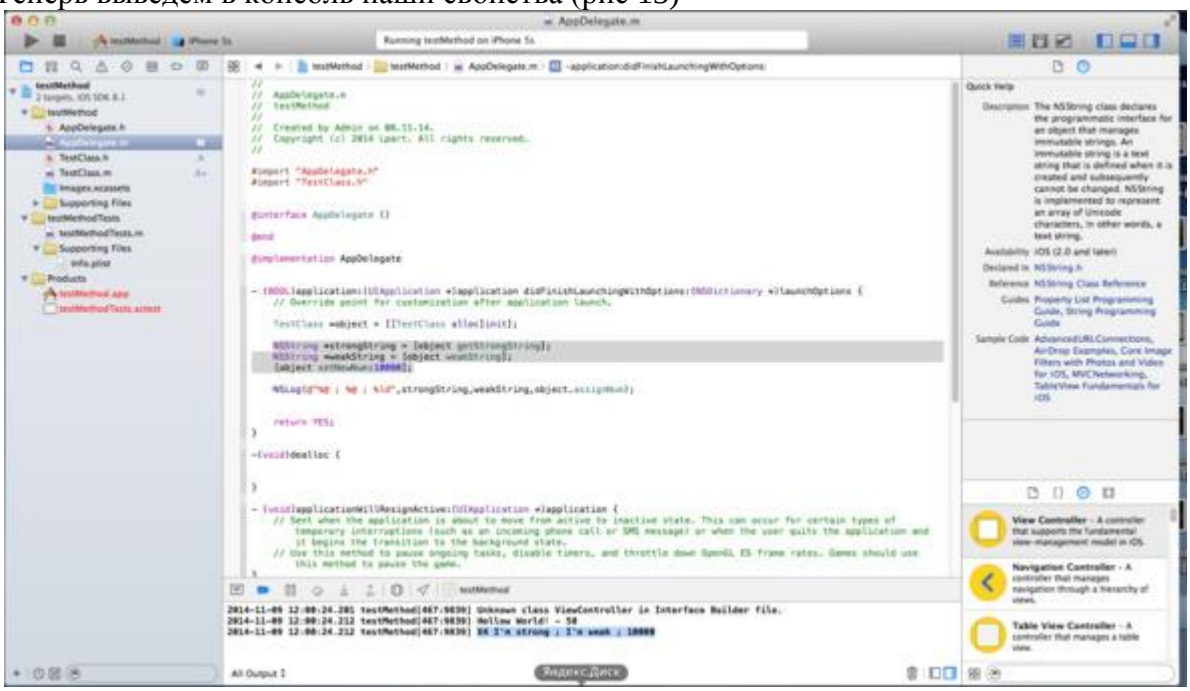


Рисунок 13 — Вывод свойств в консоль

Рассмотрим параметры property:

1) Strong — это ссылка на память, указывая на объект, не позволяет объекту быть уничтоженным, после того как обнулим эту ссылку, объект будет уничтожен.

2) Weak — ссылка, которая не держит объект, то есть он будет уничтожен, если ни где больше не будет использоваться.

3) Assign — параметр, похожий на weak с той разницы, что assign будет жить даже после уничтожения объекта. Используется для примитивных типов данных (int, float, NSInteger, CGFloat, и т.д.)

4) Atomic — создает по умолчанию свойство, защищенную от одновременного обращения потоков к ней.

5) Nonatromic — обратное atomic.

6) Copy — создает свойство, которое копирует значение, а не указатель.

7) Readonly и readwrite — первое, открывает доступ только с геттеру, второе к геттеру и сеттеру.

8) Retain — указывает на то, что управление памятью будет происходить в ручную.

Раньше работа с память в Objective-C осуществлялась в ручную, теперь используется ARC (Automatic Reference Counting), не путайте со сборщиком мусора. Это механизм автоматического подсчета ссылок, который перекладывает управление памятью на компилятор. Теперь вам не нужно освобождать объекты самим, что значительно упрощает процесс разработки и снижает риск создания утечек памяти. Компилятор полностью «понимает» когда объект создается, а когда его нужно уничтожить.

Категории

Нередко возникает ситуация, когда необходимо добавить метод уже существующему классу, не изменяя его. Objective C позволяет это сделать с помощью категорий и наследования.

Для начала создадим новый проект, выберем шаблон Single View Application.

Создадим новую категорию. Для этого выберем File -> New File -> Objective C File. Назовём её MyCategory, выберем File Type — Category, выберем класс NSString.

Рассмотрим по подробнее:

NSString+MyCategory. h
@interface NSString (MyCategory)

@end

Чем то напоминает определение класса. Отличие в том, что после директивы @interface следует имя расширяемого класса, а в скобках указывается название категории.

Добавим новый метод print, который будет выводить в лог содержание строки.

@interface NSString (MyCategory)

— (void) print;

@end

Реализуем его.

NSString+MyCategory. m
#import «NSString+MyCategory. h»

@implementation NSString (MyCategory)

— (void) print {
NSLog (@"%@», self);
}

@end

Теперь используем нашу категорию. Откроем ViewController. m и добавим в метод ViewDidLoad следующий код.

#import «NSString+MyCategory. h»
— (void) viewDidLoad {
[super viewDidLoad];
NSString *someString = @«Some String»;
[someString print];
}

Запустим приложение. В консоль выведется следующее:

2014-11-14 14:50:10.878 lesson [2947:12292] Some String

Таким образом, мы добавили новый метод классу NSString.

Категории так же позволяют переопределять методы класса, как и при наследовании, но такое недопустимо, так как для достижения таких целей необходимо применять наследование.

Добавление новых instance-переменных

Добавление к существующему классу новых переменных с помощью категории, немного отличается от добавления переменных в новый класс. Для начала объявим её.

NSString+MyCategory. h
@interface NSString (MyCategory)

@property (nonatomic, assign) int myProperty;

@end

Теперь реализуем методы доступа для новой переменной (они не генерируются компилятором автоматически), перед этим необходимо включить библиотеку objc-runtime

#import <objc/objc-runtime. h>

@implementation NSString (MyCategory)

static char kMyProperty [] = «myProperty»;

— (void) setMyProperty: (int) myProperty {
objc_setAssociatedObject (self, kMyProperty, @ (myProperty), OBJC_ASSOCIATION_ASSIGN);
}

— (int) myProperty {
id property = objc_getAssociatedObject (self, kMyProperty);
return [property intValue];
}

@end

Обратите внимание, реализация методов доступа отличается от реализации их в конкретном классе.

Рассмотрим сеттер. Функция objc_setAssociatedObject принимает параметры владельца переменной класса (self); ключа, по которому будет хранится ссылка на переменную (kMyProperty); саму переменную, id типа (примитивные переменные (int, char, struct и т.п.), необходимо явно привести к id, используя литерал @ (переменная), либо обернуть в модель (NSNumber, NSValue, и т.п.)).

В геттере используется функция objc_getAssociatedObject, которая возвращает переменную, принимая владельца (self) и ключ, по которому хранится переменная.

Проверим новую переменную.

— (void) viewDidLoad {
[super viewDidLoad];
NSString *someString = @«Some String»;
someString.myProperty = 5;
NSLog (@"%d», someString.myProperty);
}

Запустим приложение. В лог выведется следующее

2014-11-14 16:06:41.450 lesson [3378:32478] 5

Таким образом, мы добавили новую instance-переменную к классу NSString.

Делегирование

Делегирование мощный инструмент для решения многих проблем проектирования. Сущность это метода заключается в том, что некоторый метод класса реализуется не в конкретном классе, а в произвольном. Приступим.

Для начала создадим новый класс. Переопределим init метод и обьявим некий абстрактный метод (myClassMethod)

MyClass. h

@interface MyClass: NSObject

```
@property (nonatomic, weak) id <MyProtocol> delegate;

— (void) myClassMethod;
— (id) init;

@end
MyClass. m
@implementation MyClass

— (id) init {
self = [super init];
if (!self) return nil;
[self performSelector:@selector (myClassMethod) withObject: nil afterDelay:2.0f];
return self;
}

— (void) myClassMethod {
NSLog (@«This is myClassMethod»);
}

@end
```

При инициализации нашего класса будет запущен таймер, который через 2 секунды вызовет метод myClassMethod.

Используем наш класс

```
ViewController. m
#import «MyClass. h»
@implementation ViewController

— (void) viewDidLoad {
[super viewDidLoad];
MyClass *myObject = [MyClass new];
}
@end
```

Запустим приложение. Через 2 секунды после запуска в консоль выведется

2014-11-16 23:18:20.862 lesson [806:19781] This is myClassMethod

Итак, теперь попробуем реализовать этот метод в классе ViewController. Но как? Для этого и существует делегирование.

Начнём с обьявления протокола (некоторого набора методов, которые может реализовать делегат) в классе MyClass.

```
@protocol MyProtocol <NSObject>

@optional
— (void) myProtocolMethod;

@end
```

Здесь мы объявляем протокол, даём ему название MyProtocol, указываем родительский (протоколы тоже можно наследовать) протокол NSObject, обьявляем метод myProtocolMethod, директива @optional указывает на то, реализация методов, объявленных под этой директивой у делегата не является обязательным. Для обязательности реализации необходимо использовать @requied.

Теперь добавим новую переменную экземпляра MyClass, которая будет хранить ссылку на делегата.

@property (nonatomic, weak) id <MyProtocol> delegate;

Обратим внимание, что ссылка должна быть слабой, при этом она должна указывать на протокол.

Теперь реализуем метод myClassMethod заново.

```
— (void) myClassMethod {
if ([self.delegate respondsToSelector:@selector (myProtocolMethod)]) {
[self.delegate myProtocolMethod];
}
}
```

Разберёмся, что же тут происходит. Мы пытаемся вызвать у делегата метод myProtocolMethod, при этом проверяем, реализован ли этот метод у делегата.

Теперь реализуем этот метод у делегата. Для начала необходимо принять протокол.

```
ViewController. m
@interface ViewController: UIViewController <MyProtocol>
```

Теперь в методе ViewDidLoad добавим следующий код.

```
— (void) viewDidLoad {
[super viewDidLoad];
MyClass *myObject = [MyClass new];
myObject.delegate = self;
}
```

Здесь мы создаем новый экземпляр класса MyClass, и назначаем себя делегатом.

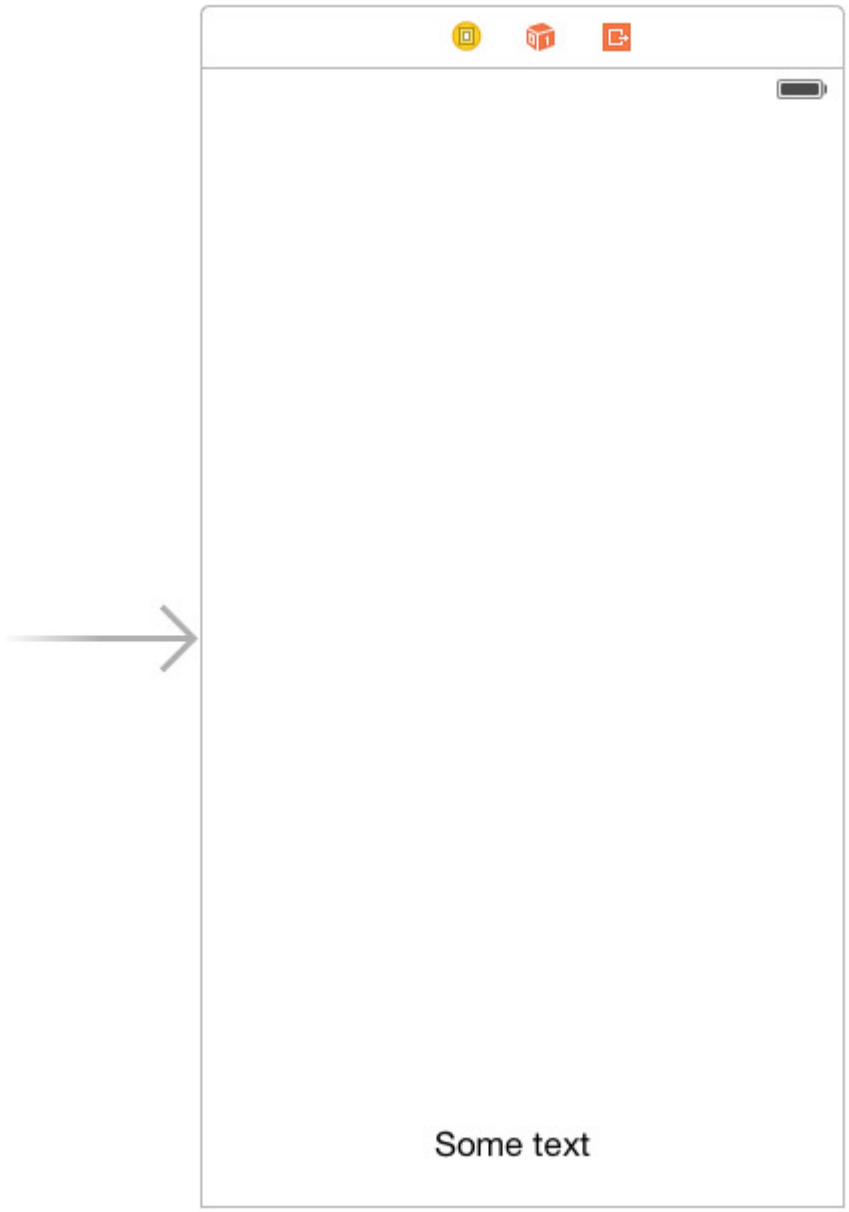
Осталось реализовать метод myProtocolMethod

```
— (void) myProtocolMethod {
NSLog (@«This method works!!»);
}
```

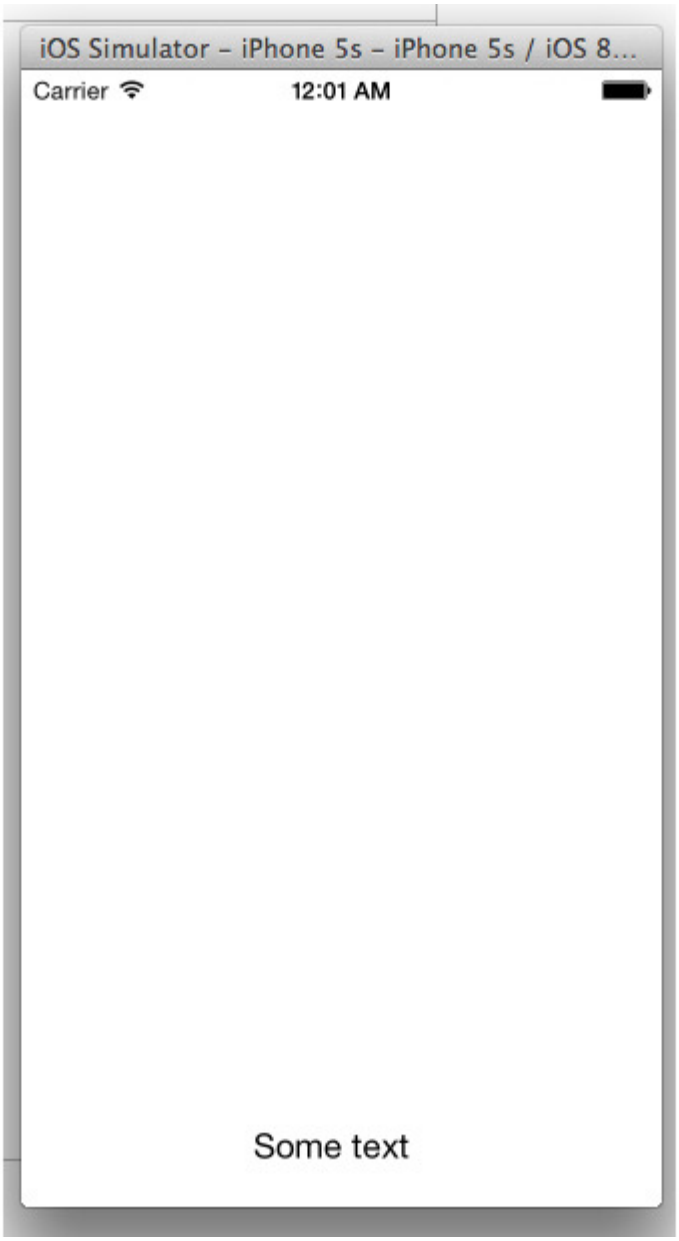
Запустим приложение, через 2 секунды после запуска мы консоль выведется следующее
2014-11-16 23:18:20.862 lesson [806:19781] This method works!!

ВЕРСТКА. Autolayout

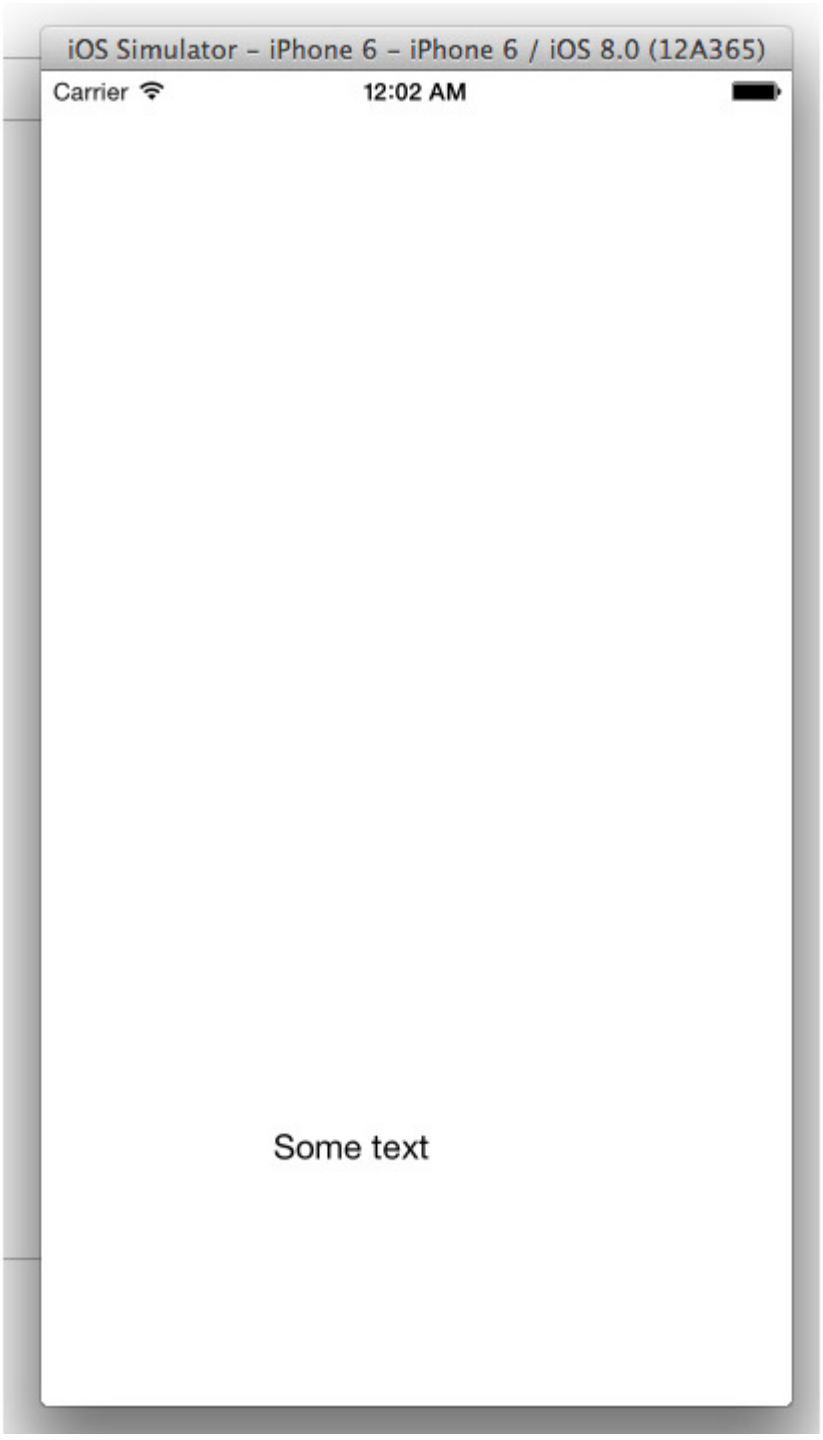
Часто при верстке экранов для мобильного приложения возникает проблема адаптации под все виды устройств. Рассмотрим типичную проблему.

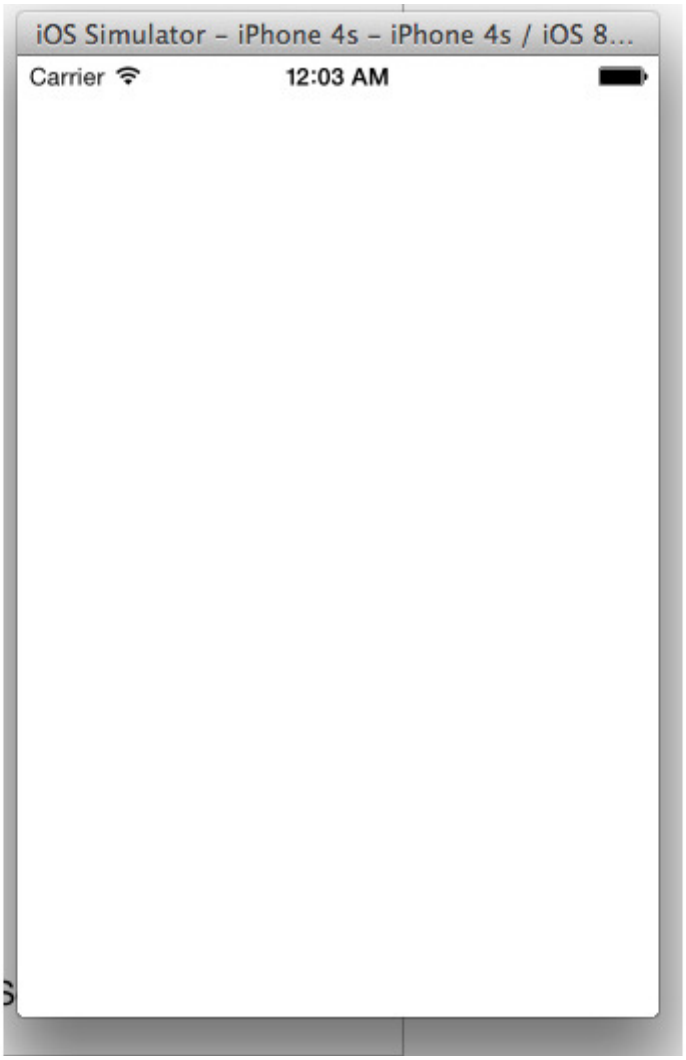


Здесь на экране UILabel. Попробуем запустить на iPhone 5.



Здесь вроде бы всё нормально, как и задумывалось. Теперь посмотрим, как он будет отображатся на iPhone 6 и iPhone 4.


iOS Simulator – iPhone 6 – iPhone 6 / iOS 8.0 (12A365)
Carrier
12:02 AM
Some text



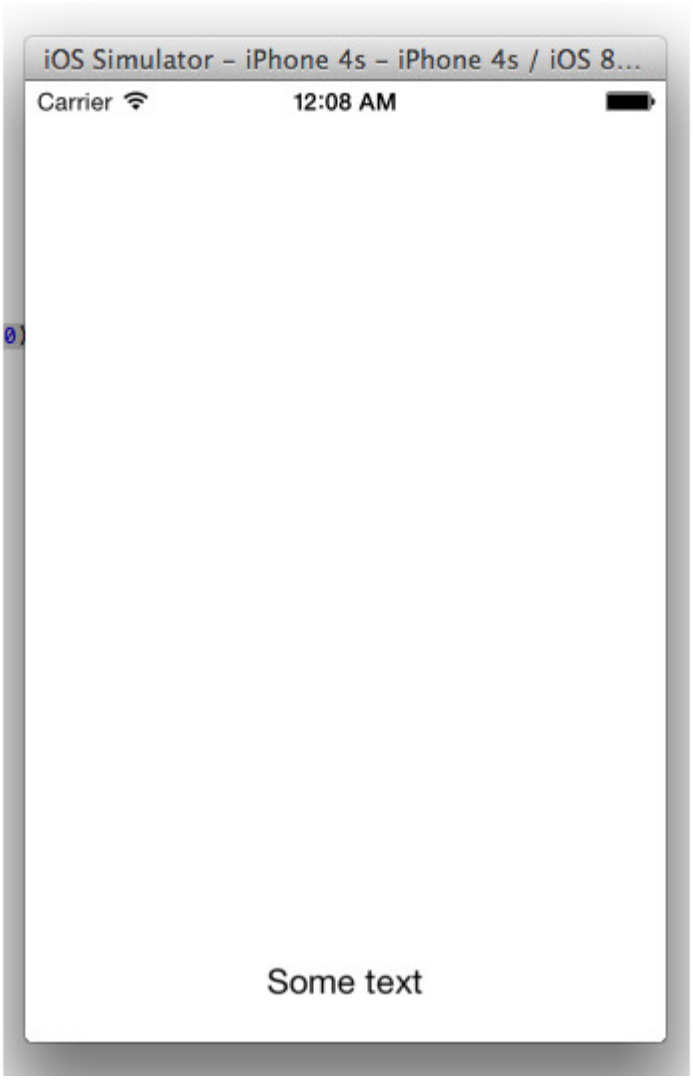
На шестом лейбл не по центру, да и снизу промежуток больше, чем нужно.

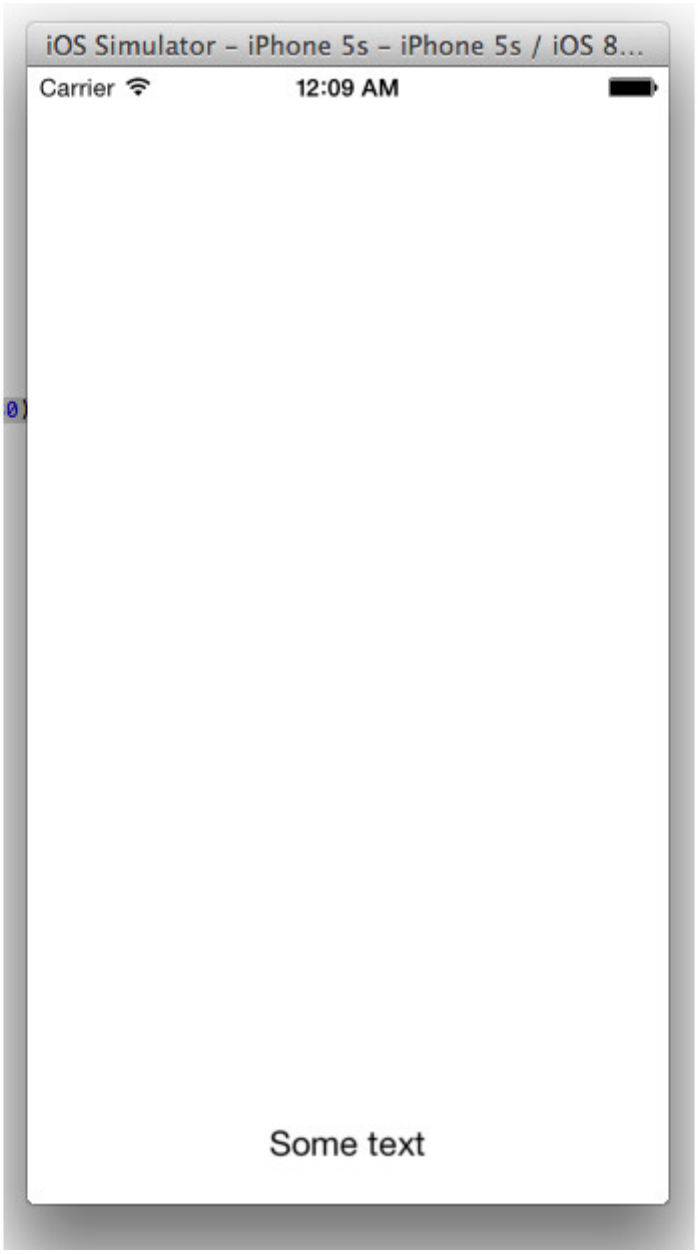
А на четвёртом и вовсе его нет. Как же быть?

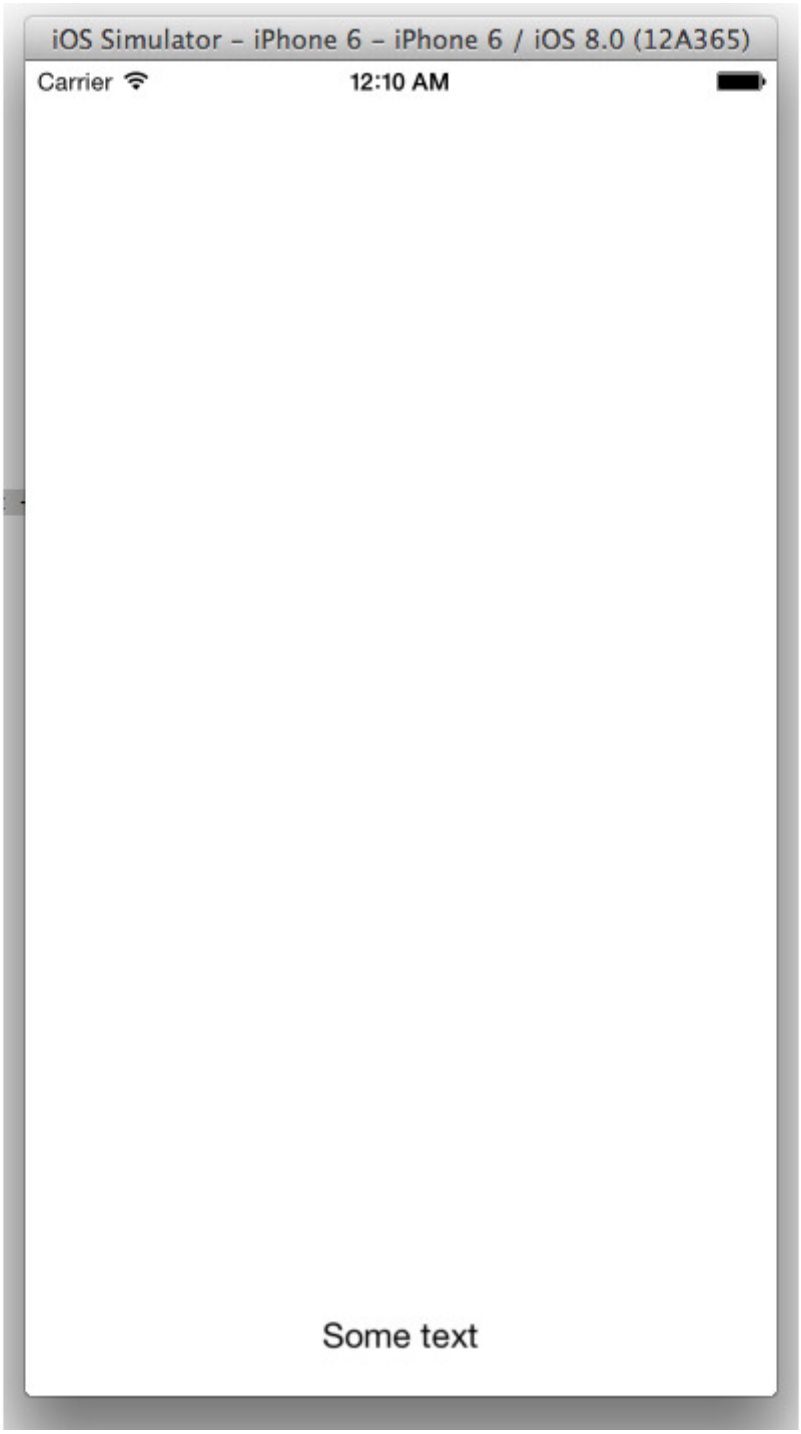
Можно положение лейбла на экране посчитать программно, например так:

self.label. center = CGPointMake(self.view. center. x, self.view.frame.size. height — 30);

Здесь мы задаём координаты центра UILabel — центр экрана по x, и отступ снизу по y. И действительно, это работает.

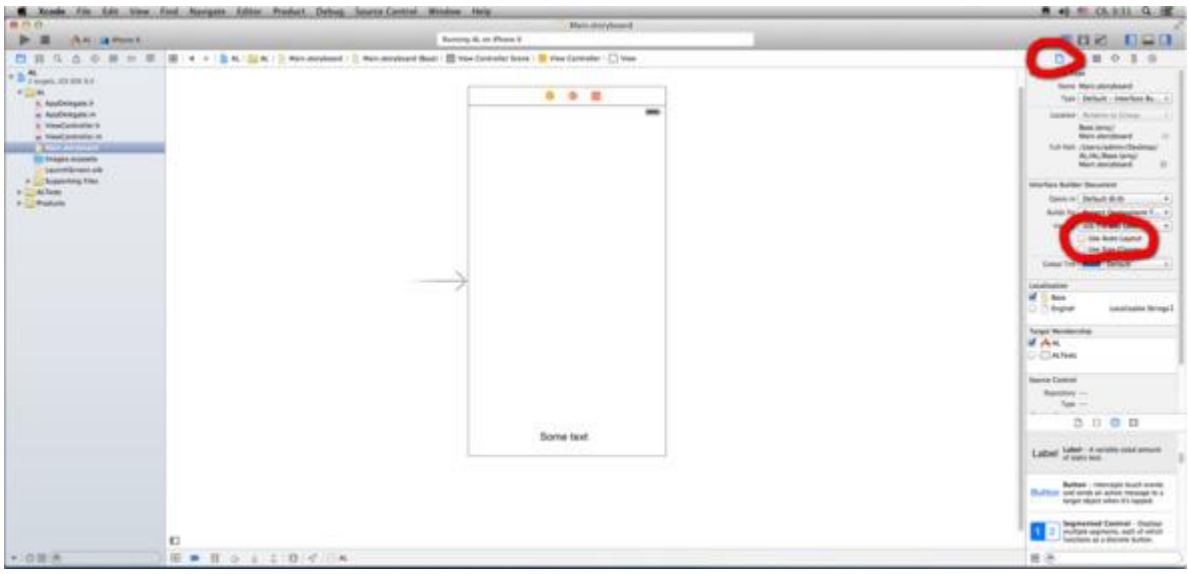

iOS Simulator – iPhone 4s – iPhone 4s / iOS 8...
Carrier
12:08 AM
Some text


iOS Simulator – iPhone 5s – iPhone 5s / iOS 8...
Carrier
12:09 AM
Some text

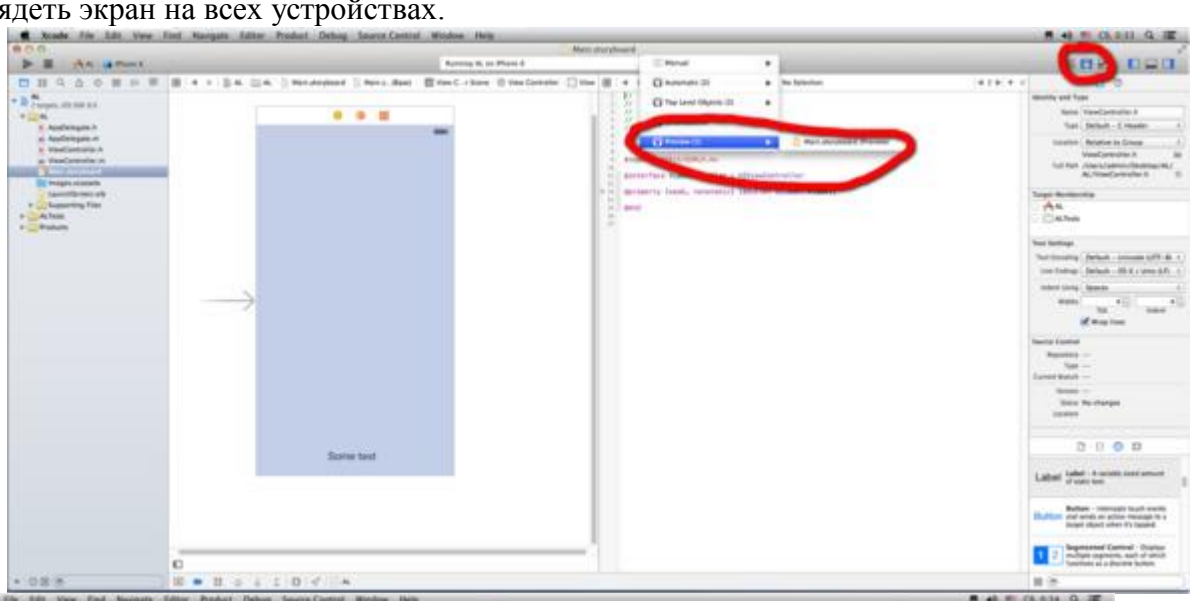


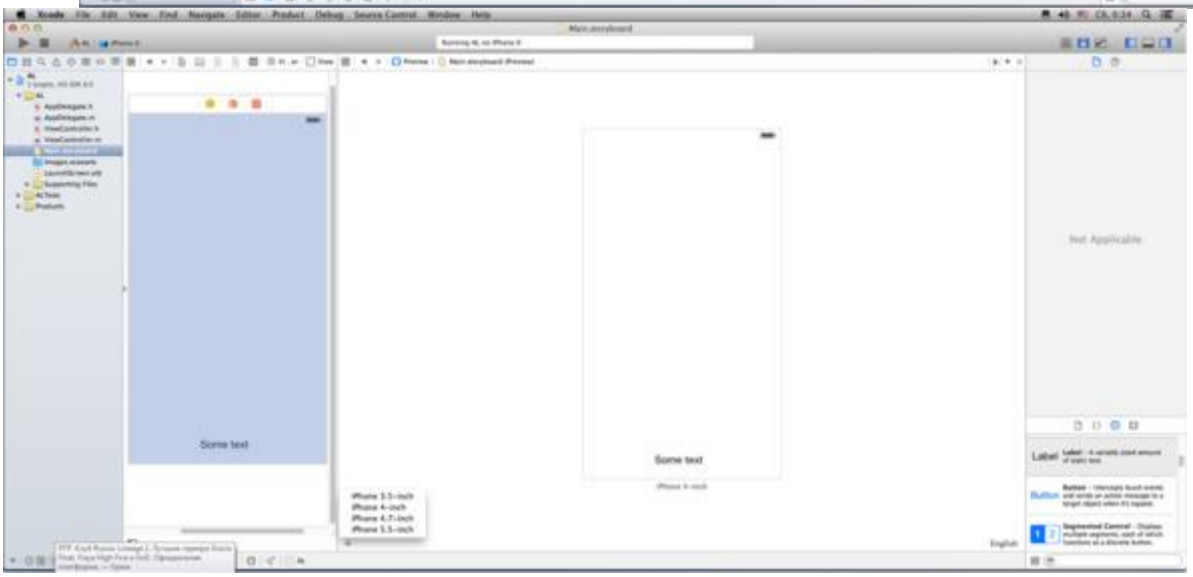
Но такой способ очень неудобен: во первых — пропорции положения необходимо задавать вручную кодом, без использования Interface Builder. Это значит, что задавая фрейм компонента, необходимо заново компилировать приложение, и смотреть, как это будет выглядеть на целевом устройстве. Во вторых — такой метод слишком нагромождает код, вместо логики работы приложения: логика отрисовки UI. Да и при большом количестве элементов UI на экране писать для каждого логику отрисовки довольно рутинно.

Для решения подобных проблем, xCode предоставляет функцию AutoLayout. По умолчанию эта опция включена, если нет — то включим её. Выберем вкладку Idenity and type, и поставим галочку напротив Use Auto Layout.

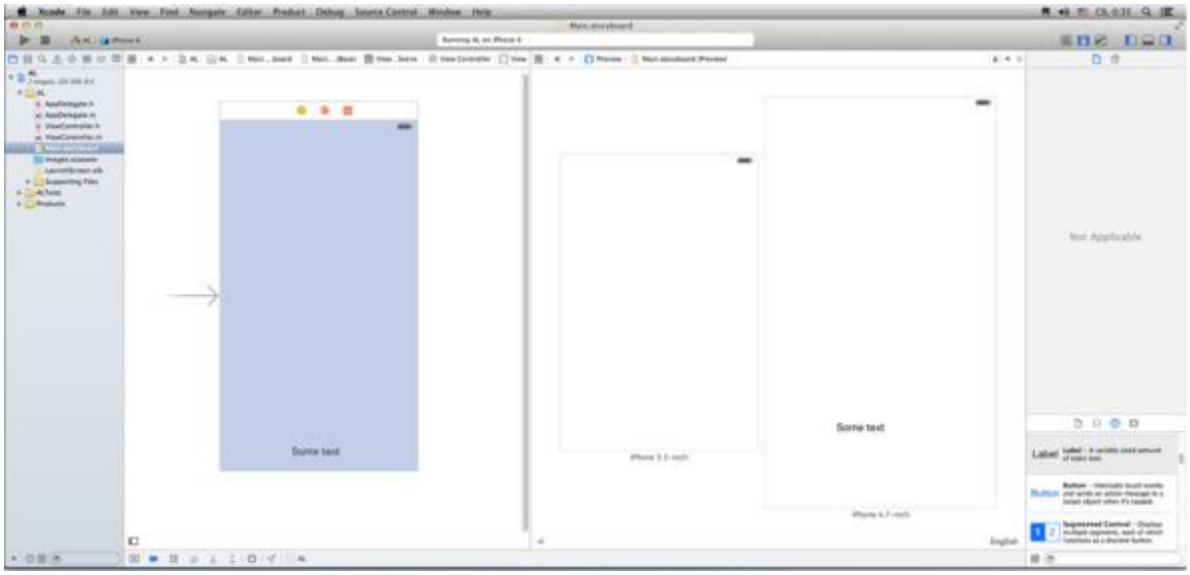


В xCode 6 появилась удобная функция Preview — с её помощью можно видеть, как будет выглядеть экран на всех устройствах.





Добавим несколько экранов (3,5» — iPhone 4 и 4,7» = iPhone 6).

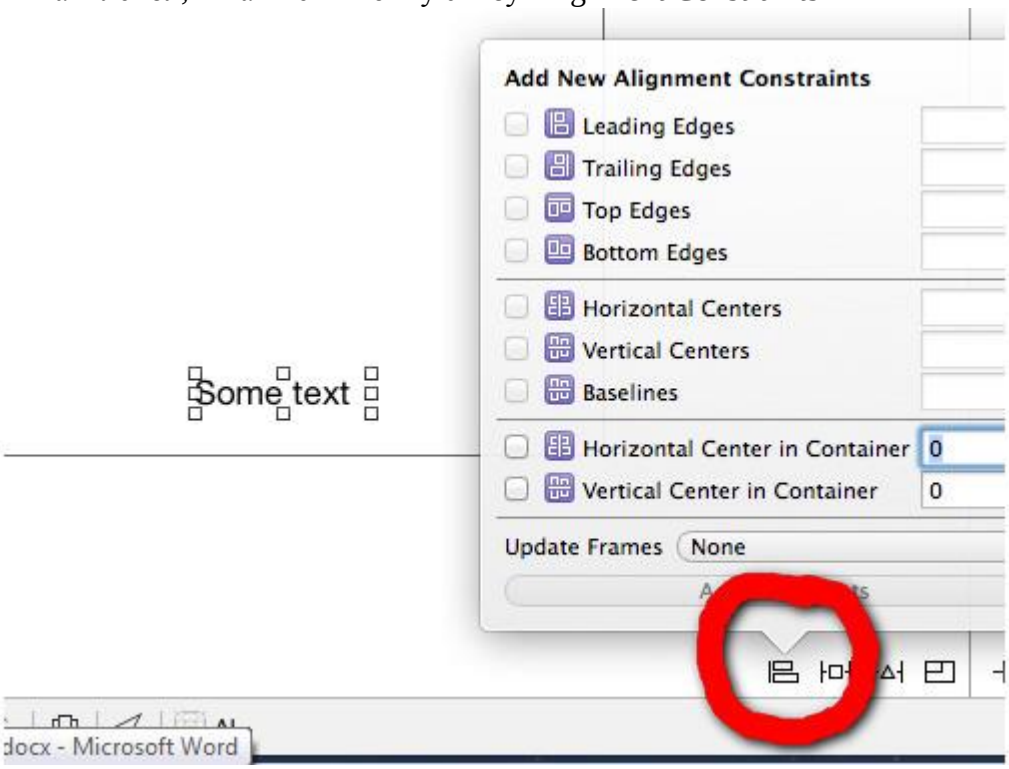


Constraints

Constraints — это величины, описывающие геометрическое положение элемента на экране по отношению к другому элементу.

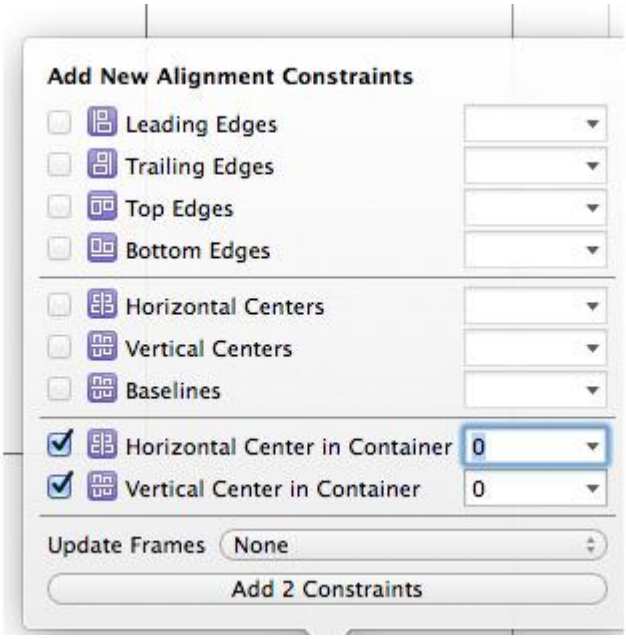
Alignment constraints

Выделим наш лейбл, и нажмём кнопку снизу Alignment Constraints



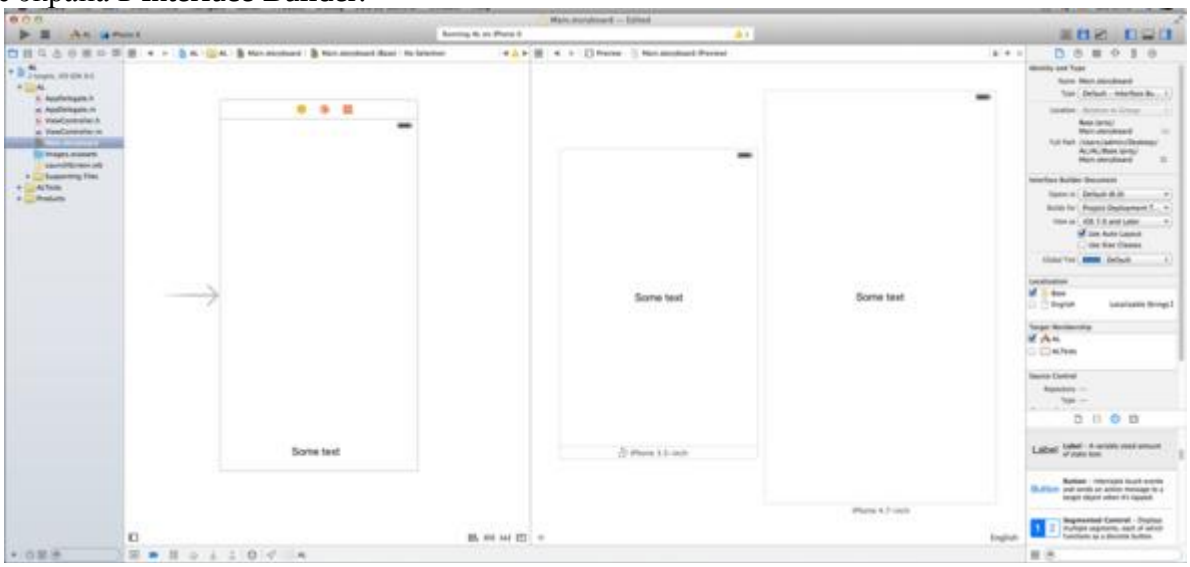
Для одиночного элемента предоставляется 2 constraints — положение по центру в родительском view по координатам x или y. При этом можно задать смещение.

Зададим нашему лейблу центрирование в родительской view. Для этого выделим 2 пункта



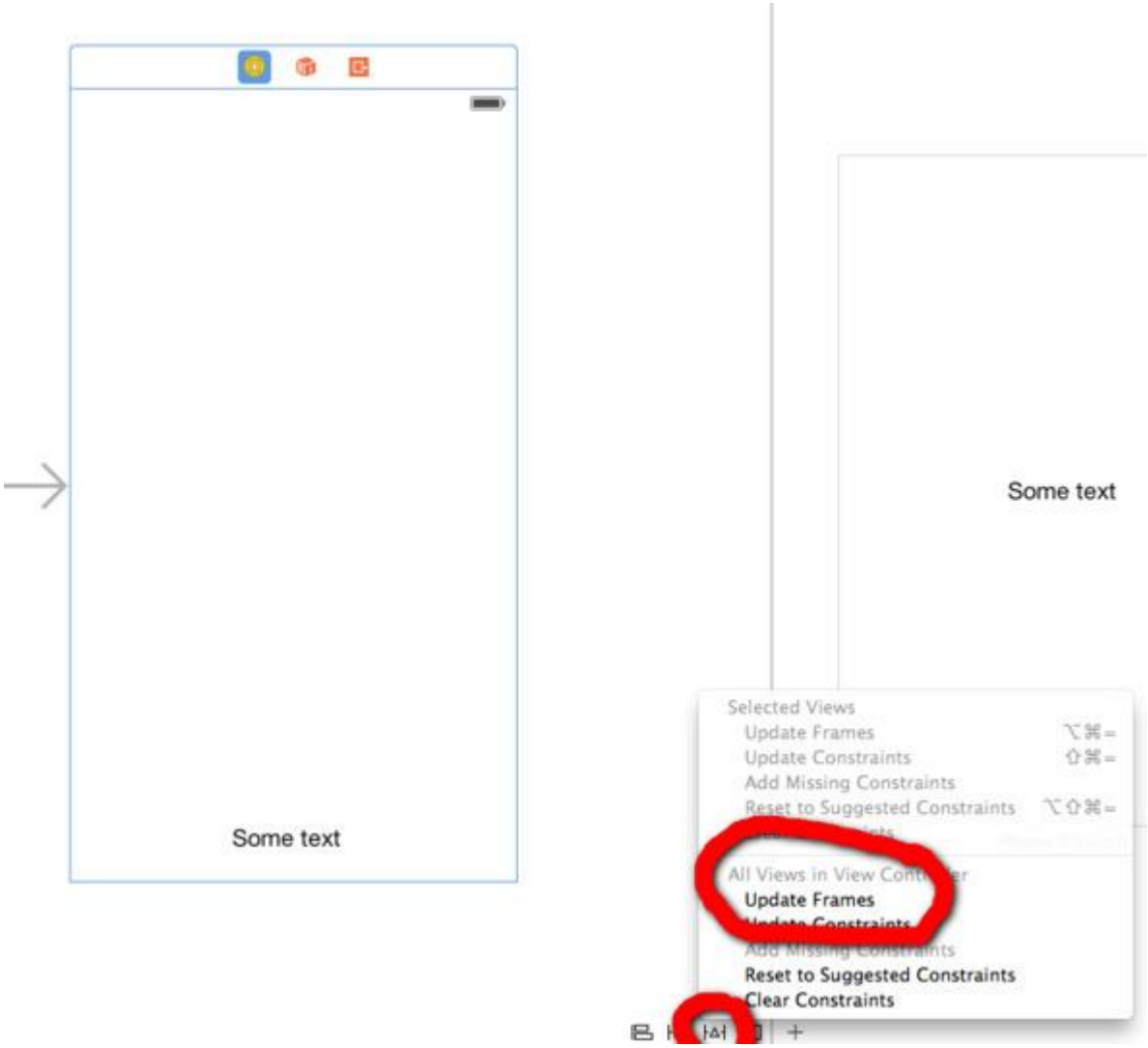
И нажмём Add 2 Constraints.

Обратите внимание. Теперь в Preview лейбл стоит строго по центру на всех экранах, кроме экрана в Interface Builder.



Для того, чтобы увидеть изменения в Interface Builder, необходимо обновить фреймы

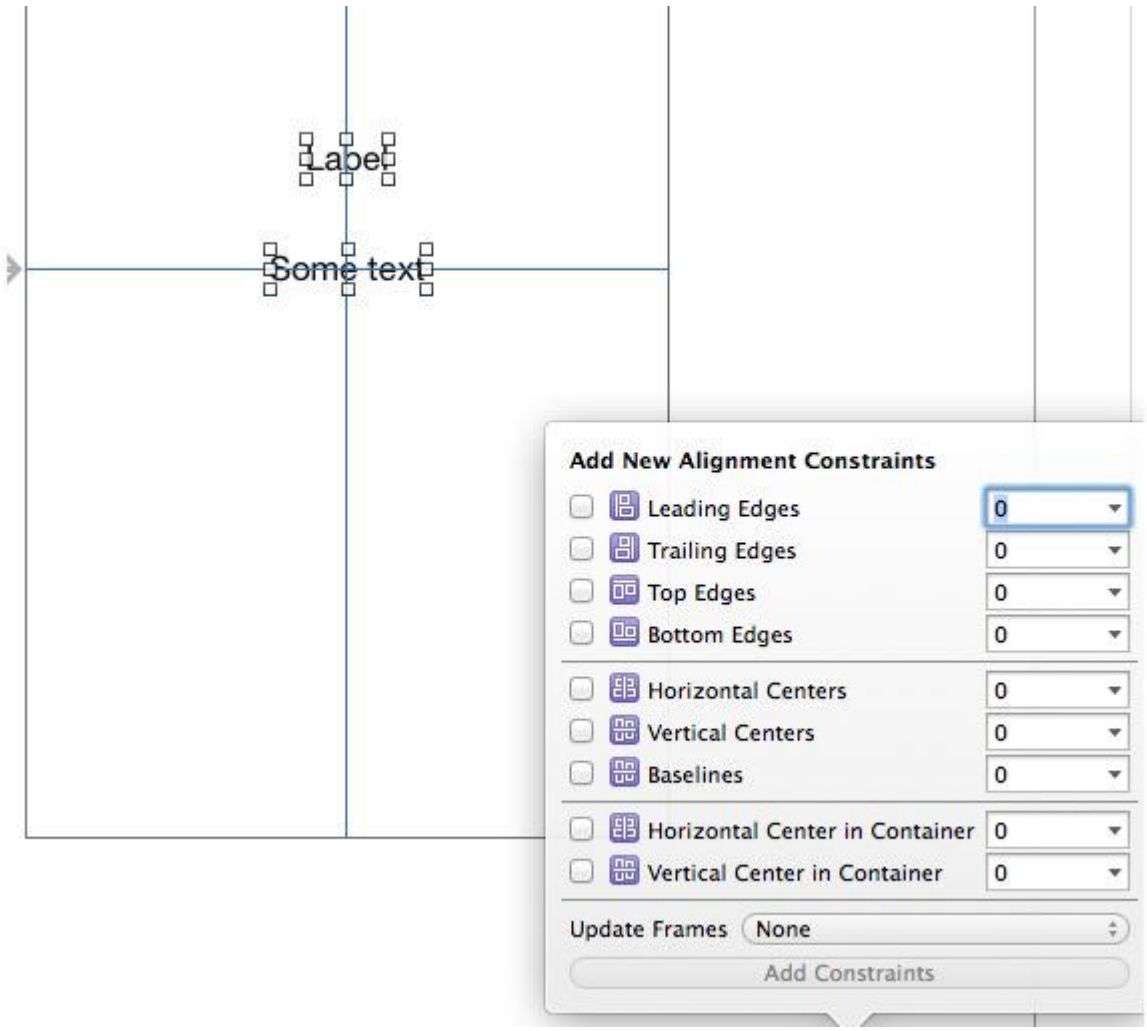
Нажмём на кнопку и выберем пункт «Update frames»



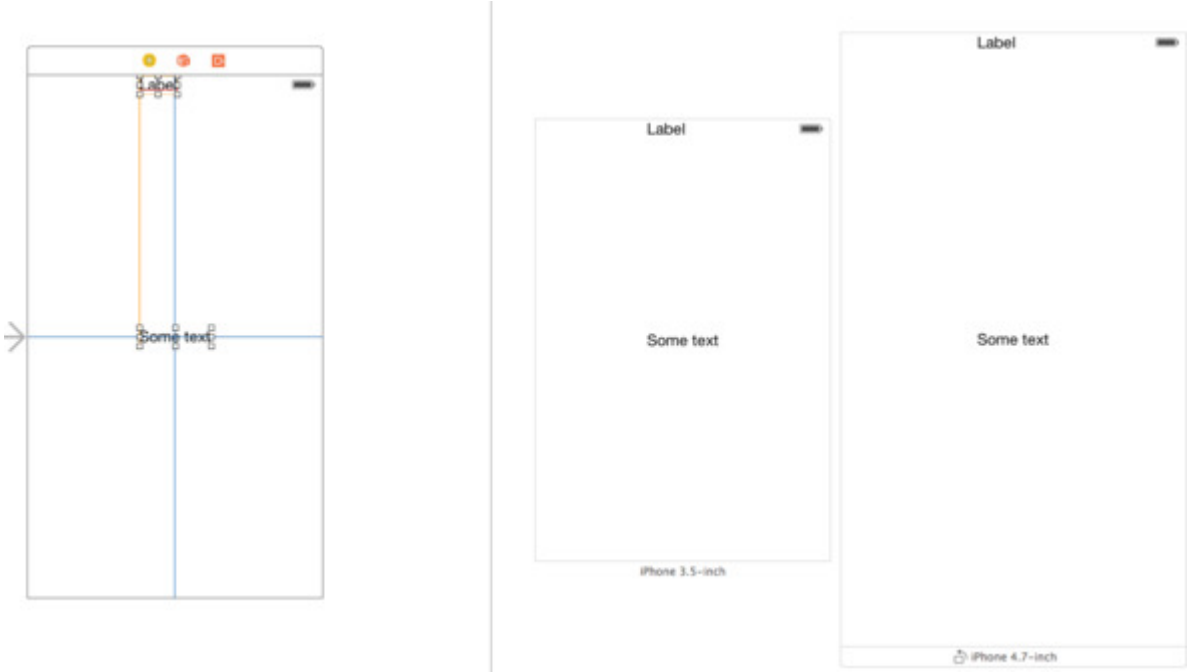
Замечание. Для корректного отображения положения необходимы Constraints по x и y. В случае отсутствия, будет выбрано величина, равная нулю. То же касается и размеров (о них далее). Но, для большинства элементов, при отсутствии заданного размера, он будет подбираться автоматически, исходя из размера содержимого. Так например, размер UILabel будет исходя от длины текста. И самое главное — фрейм элемента, остаётся неизменным, и не влияет на видимый фрем.

Рассмотрим другие пункты. Добавим ещё один UILabel.

Выделим оба лейбла. Теперь нам доступны все виды Alignment constraints.



Первые 4 — это положения краёв, относительно друг друга. Для примера, выберем Leading edges.



Второй UILabel теперь находится наверху, поскольку мы не задавали constraint для y. Но можно заметить, что координаты x обоих лейблов одинаковы.

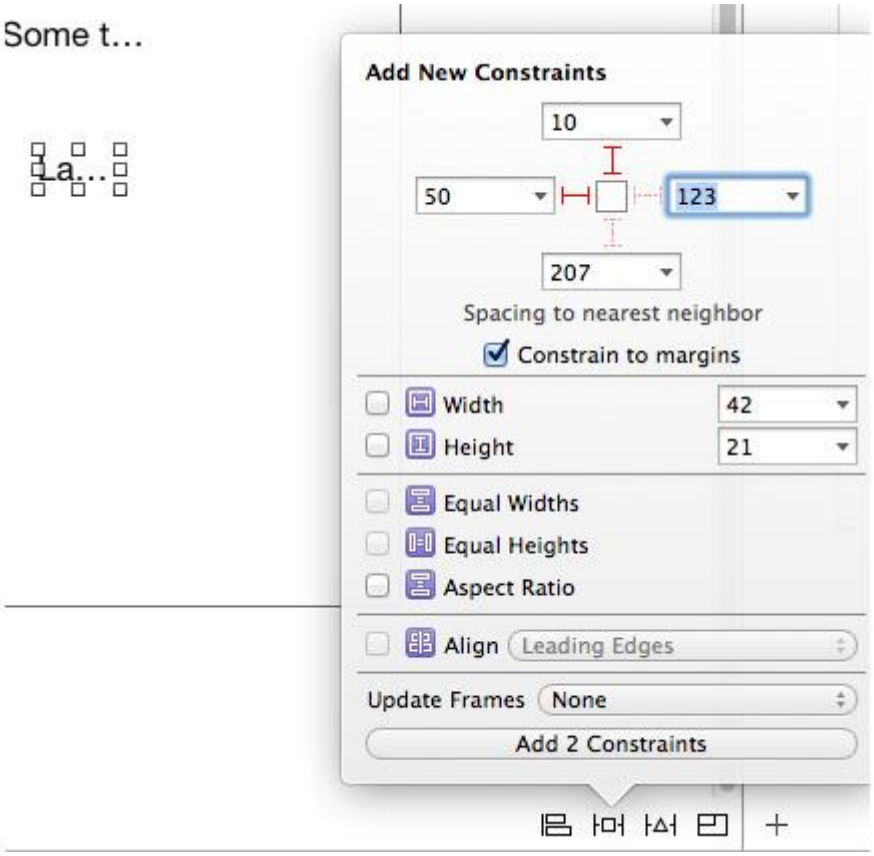
Пункты Horizontal и Vertical centers — выравнивание элементов относительно друг друга по x и y соответственно.

Последние 2 уже рассматривались — выравнивание центра относительно контейнера (родительского view).

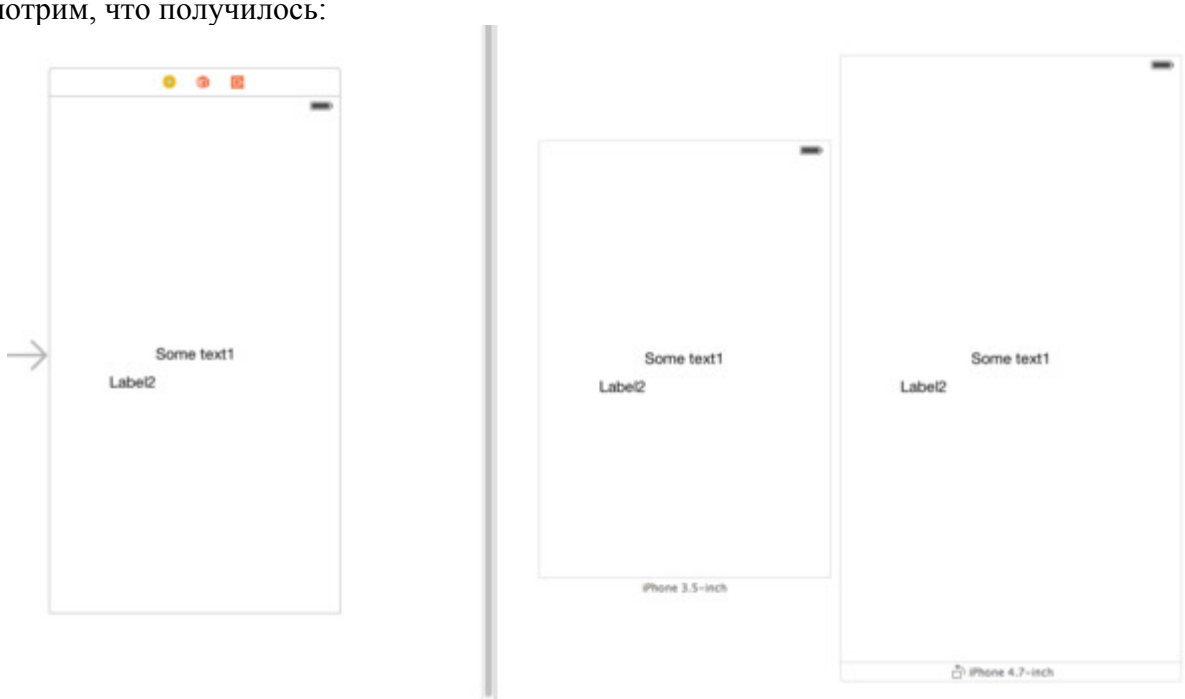
Pin

Здесь можно задать положение до ближайшей View, а так же задать размеры (фиксированные, либо равные ближайшей view).

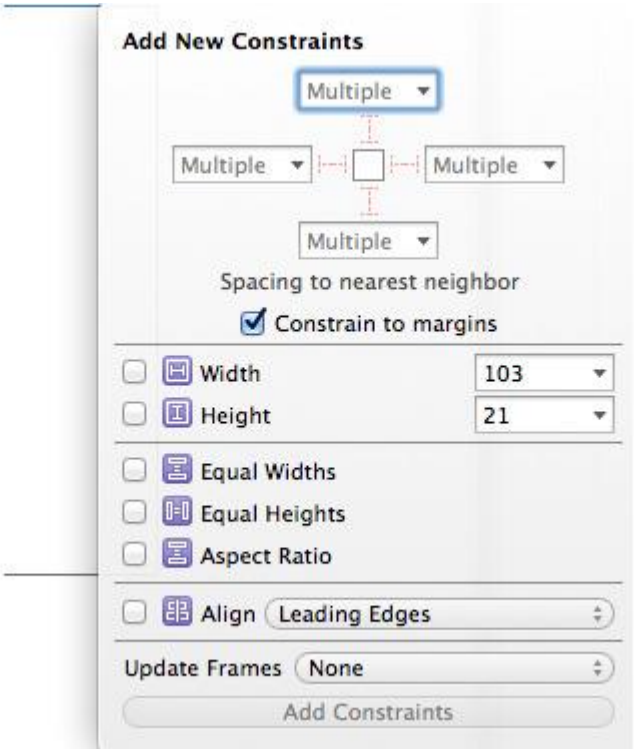
Зададим лейбл по центру. А второму назначим расстояние до первого 10 по y, и 50 от края ViewController по x.



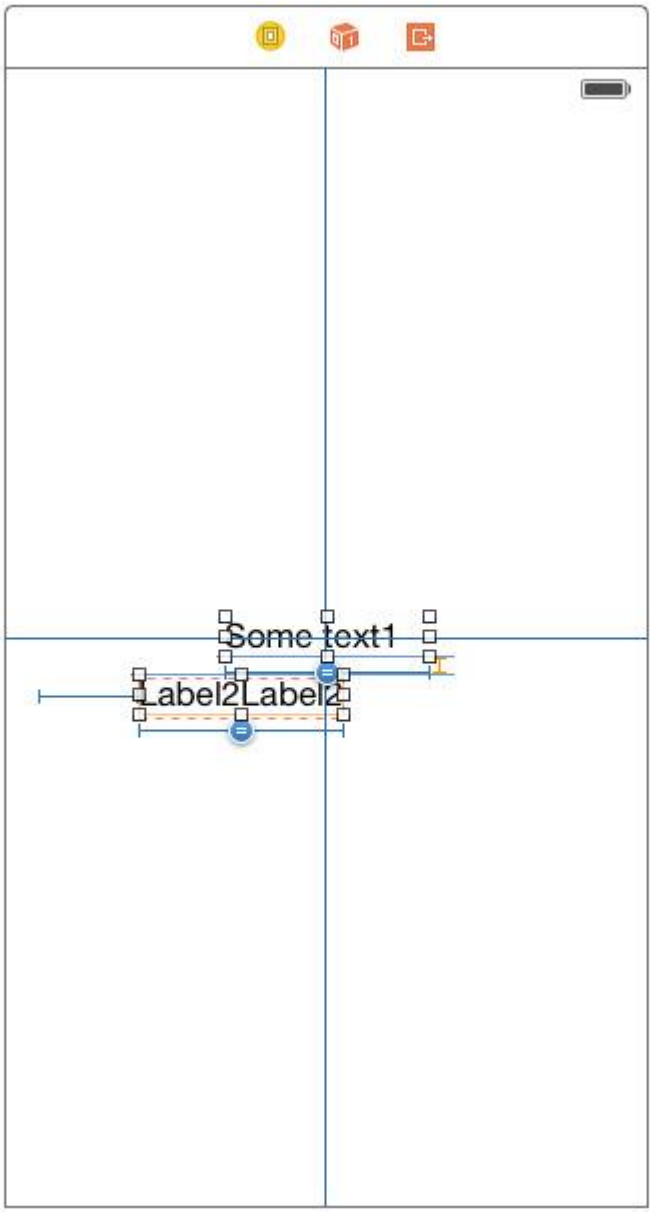
Посмотрим, что получилось:



Теперь зададим обоим равную длину. Выделим оба лейбла, выберем Equal Widths

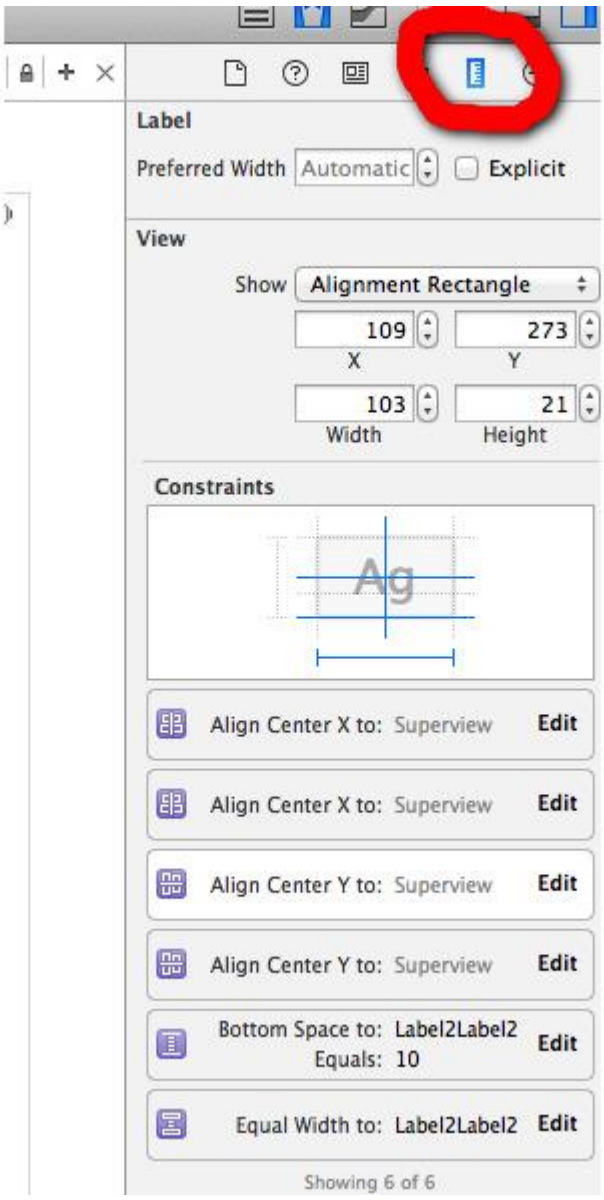


Посмотрим, что получилось

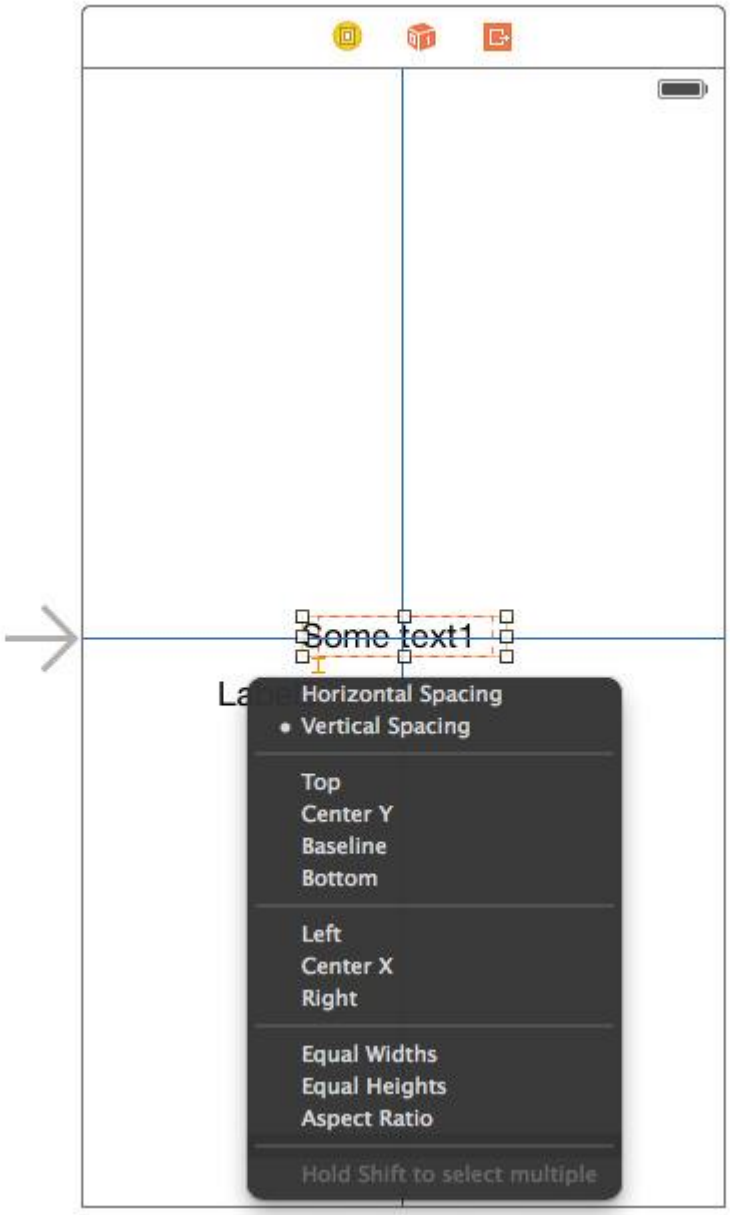


Замечу, в случае, если длины двух элементов привязанны друг к другу, то в качестве эталонной будет выбираться та, которая заданна constraint'ом. В противном случае будет выбираться наибольшая.

Было упомянуто «привязанны друг к другу». Что это? Это значит, что constraint одного элемента зависит от другого. В самом начале мы устанавливали программно зависимость положения UILabel о размера родительской View. И с AutoLayout так же. Увидеть зависимости constraint'ов можно в Size Inspector.



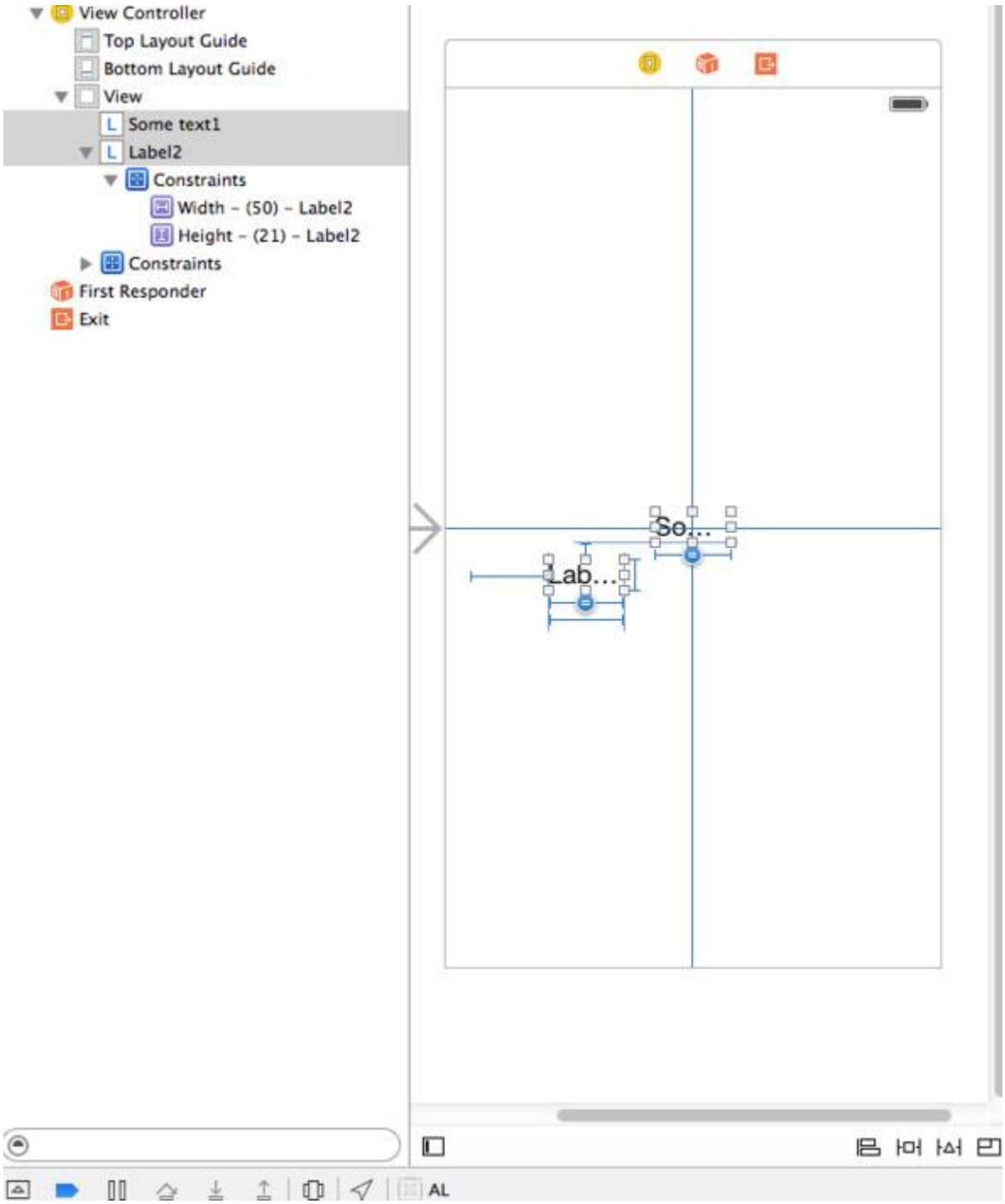
Альтернативный способ добавления constraint’a: необходимо правой кнопкой мыши выбрать первую view, и не отпуская, довести до второй.



Здесь так же можно выбрать нужные constraints, для множественного выбора нужно зажать клавижу Shift, когда всё нужное выбрано — Enter. Причём, при выборе horizontal/vertical spacing и тому подобных в качестве констаты будет выбранно текущее расстояние/размер и т. п.

Так же можно на constraint можно повесить IBOutlet, процесс добавления тот же, что и компонентов UI.

@property (weak, nonatomic) IBOutlet NSLayoutConstraint *widthConstraint;



Теперь можно программно манипулировать значением, удалять её и т. д. Например, изменение значения можно сделать так:

```
self.widthConstraint.constant = 50;
```

Или удалить со view:

```
[self.label removeConstraint: self. widthConstraint];
```

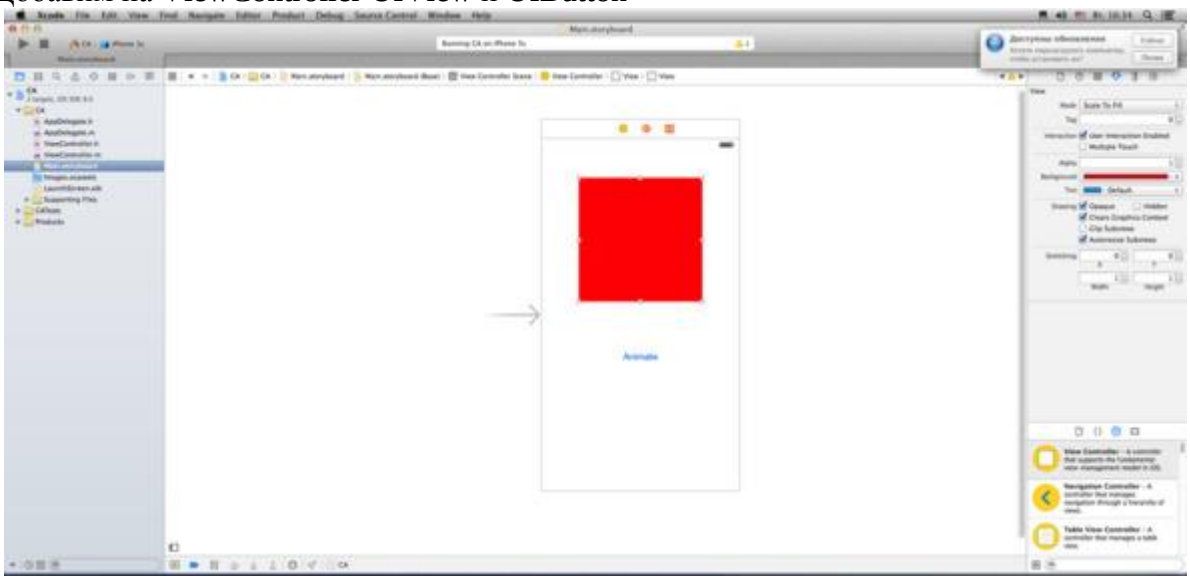
Core Animation

Core Animation — технология, разработанная компанией Apple Inc. для создания анимаций. Apple впервые публично продемонстрировала её 7 августа 2006 года во время выступления Worldwide Developers Conference. Core Animation выполняется на отдельном

от основных программ потоке, практически не влияя на производительность системы на многоядерных машинах. Однако данная технология требует Core Image-совместимого Mac.

Анимации с Core Animation автоматизированы и могут быть получены с минимальным вмешательством разработчика. Когда разработчик изменяет атрибут компонента, Core Animation автоматически видоизменяет его промежуточными шагами (цвет, прозрачность и т. д.) между начальным и конечным значениями, визуально улучшая приложения и уменьшая количество исходного кода, который бы потребовался при использовании стандартных средств и технологий, предоставляемых Cocoa.

Для начала создадим новый проект, выберем шаблон Single View Application.
Добавим на ViewController UIView и UIButton



Создадим аутлет для UIView

@property (weak, nonatomic) IBOutlet UIView *rectView;

А также IBAction для UIButton

— (IBAction) animate {
}

Простые анимации

Основой CoreAnimation является класс CALayer, который является родительским классом CoreAnimation.

Как заявляет Apple — CoreAnimation это 2.5D технология. Что оно из себя представляет? Есть 3D пространство, а есть слои 2D объекты, которые можно перемещать в 3D пространстве, по трём осям. Немного заглядывая вглубь — на самом деле, UIView и все его подклассы, являются «обёрткой» над классом CALayer.

Рассмотрим простую анимацию масштабирования. Как уже было сказано выше, CoreAnimation позволяет создавать анимации с минимальным вмешательством разработчика. Для того, чтобы проанимировать масштабирование какого то UIView, необходимо лишь задать начальные и конечные координаты и длительность анимации, всё остальное CoreAnimation сделает за вас.

— (IBAction) animate {
CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:@«transform»];

anim.fromValue = [NSValue valueWithCATransform3D: CATransform3DMakeScale (1, 1, 1)];

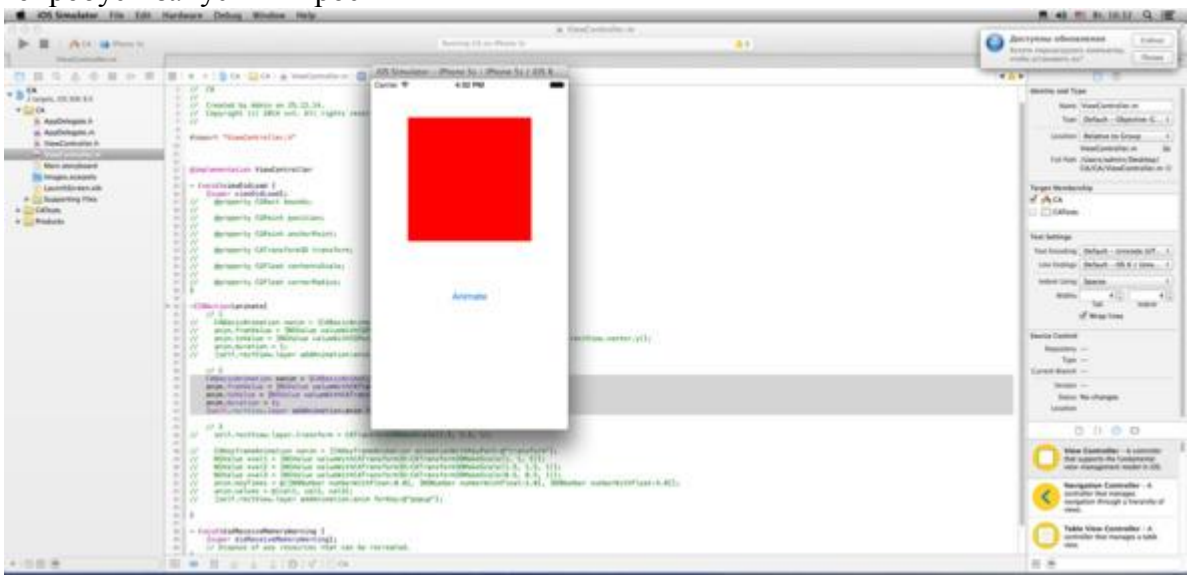
anim.toValue = [NSValue valueWithCATransform3D: CATransform3DMakeScale (1.5, 1.5, 1)];

anim. duration = 1;

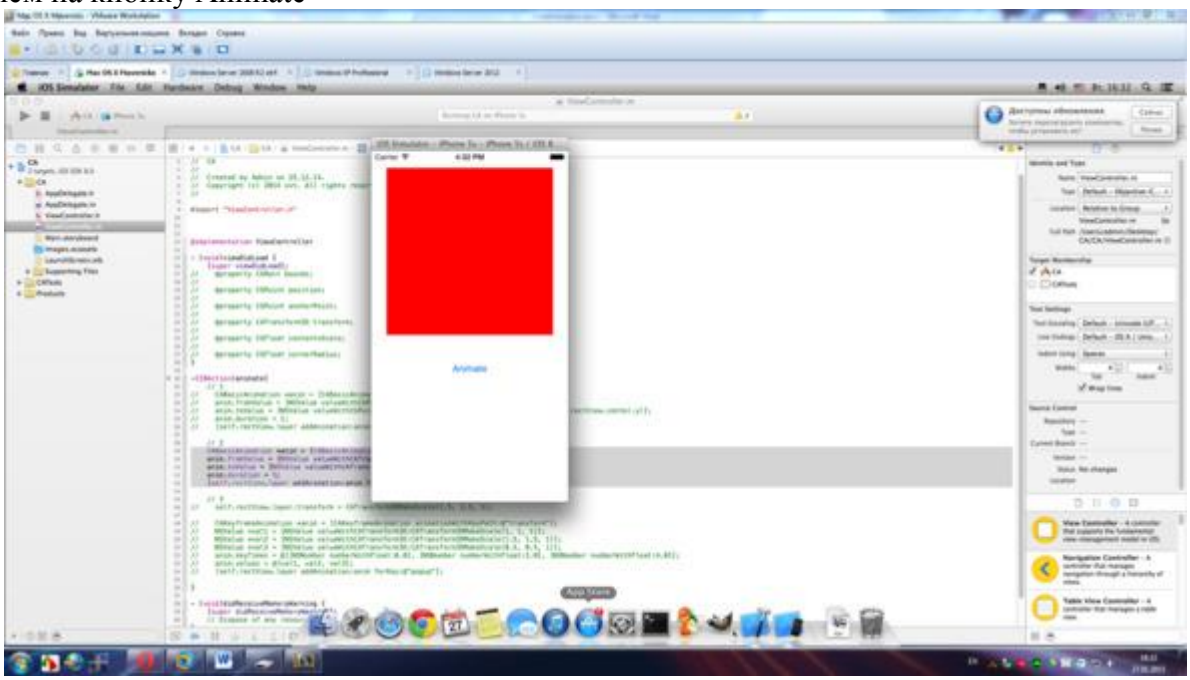
[self.rectView.layer addAnimation: anim forKey:@«popup»];

}

Попробуем запустить проект



Нажмём на кнопку Animate



Можно увидеть, как квадрат плавно увеличивается, потом возвращается в первоначальное состояние.

Разберёмся, как это работает.

CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:@«transform»];

Здесь мы создаем новый объект класса CABasicAnimation. Но что такое KeyPath? Это свойство слоя, которое будет анимироваться. Вот некоторые свойства CALayer, которые можно анимировать.

@property CGRect bounds;

@property CGPoint position;

@property CGPoint anchorPoint;

@property CATransform3D transform;

@property CGFloat contentsScale;

@property CGFloat cornerRadius;

Посмотреть, какие свойства можно анимировать, можно в хедере CALayer. h
Такое свойство будет иметь пометку «Animatable»

@property CGPoint position;

/* The Z component of the layer's position in its superlayer. Defaults
* to zero. Animatable. */

Стоит отметить, что свойство, которое вы выбираете для анимации, при её объявлении не проверяется компилятором на валидность, поэтому сделав опечатку можно получить исключение при работе приложения. Например:

CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:@«transfomr»];

Стоит заострить внимание именно на @property CATransform3D transform;

Используя его, можно получить самые различные анимации, и другие @property слоя так или иначе используют его. CATransform3D это struct вида

```
struct CATransform3D
{
CGFloat m11, m12, m13, m14;
CGFloat m21, m22, m23, m24;
CGFloat m31, m32, m33, m34;
CGFloat m41, m42, m43, m44;
};
```

Это матрица преобразований. Но для выполнения различных матричных преобразований нужны хорошие знания математики и Apple предоставляет набор функций

CATransform3D CATransform3DMakeScale (CGFloat sx, CGFloat sy, CGFloat sz)

CATransform3D CATransform3DMakeRotation (CGFloat angle, CGFloat x, CGFloat y, CGFloat z)

CATransform3D CATransform3DTranslate (CATransform3D t, CGFloat tx, CGFloat ty, CGFloat tz)

Эти и другие функции позволяют выполнить необходимые преобразования легко и просто.

Таким образом, мы присваем начальное, и конечное значение анимации

anim.fromValue = [NSValue valueWithCATransform3D: CATransform3DMakeScale (1, 1, 1)];

anim.toValue = [NSValue valueWithCATransform3D: CATransform3DMakeScale (1.5, 1.5, 1)];

Можно обратить внимание, что координаты задаются в 3х измерениях — x, y, z. Как уже и было сказано выше: CoreAnimation это 2.5D технология, в которой слои (2D объекты) можно трансформировать в трёх измерениях. Важно понимать это, и не путать с настоящими 3D объектами (для их построения и манипуляции необходимо использовать OpenGL).

Таким образом, мы задали начальные и конечные размеры слоя.

Теперь необходимо задать время выполнения анимации, в секундах.

anim. duration = 1;

Напоследок, необходимо прикрепить анимацию к слою UIView:

[self.rectView.layer addAnimation: anim forKey:@«popup»];

Здесь фигурирует некий Key (в нашем случае «popup») — это ключ, по которому можно обратиться к анимации. Можно прикрепить несколько анимаций, и обращаться к ним во время их выполнения (об этом позже).

Можно заметить, что при завершении анимации, объект на экране возвращается к своему первоначальному состоянию. Для того, что бы сохранить состояние объекта по завершению анимации, необходимо явно присвоить новое значение свойству слоя.

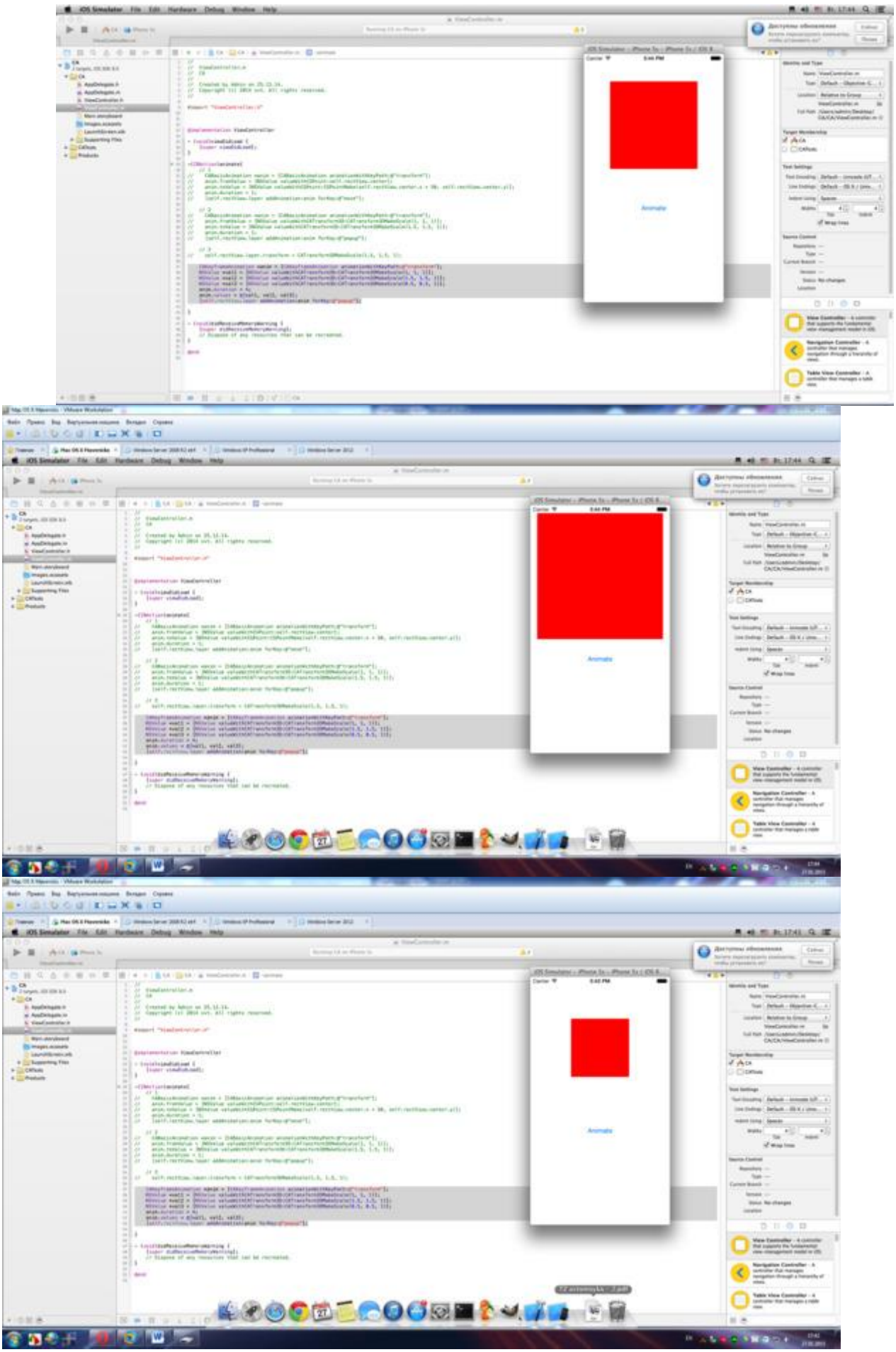
self.rectView.layer.transform = CATransform3DMakeScale (1.5, 1.5, 1);

Key-Frame анимация

Очень похожа на простую, разница в том, что вместо начального и конечного значения, используется массив значений.

CAKeyframeAnimation *anim = [CAKeyframeAnimation animationWithKeyPath:@«transform»];
NSValue *val1 = [NSValue valueWithCATransform3D: CATransform3DMakeScale (1, 1, 1)];
NSValue *val2 = [NSValue valueWithCATransform3D: CATransform3DMakeScale (1.5, 1.5, 1)];
NSValue *val3 = [NSValue valueWithCATransform3D: CATransform3DMakeScale (0.5, 0.5, 1)];
anim. duration = 4;
anim.values = @ [val1, val2, val3];
[self.rectView.layer addAnimation: anim forKey:@«popup»];

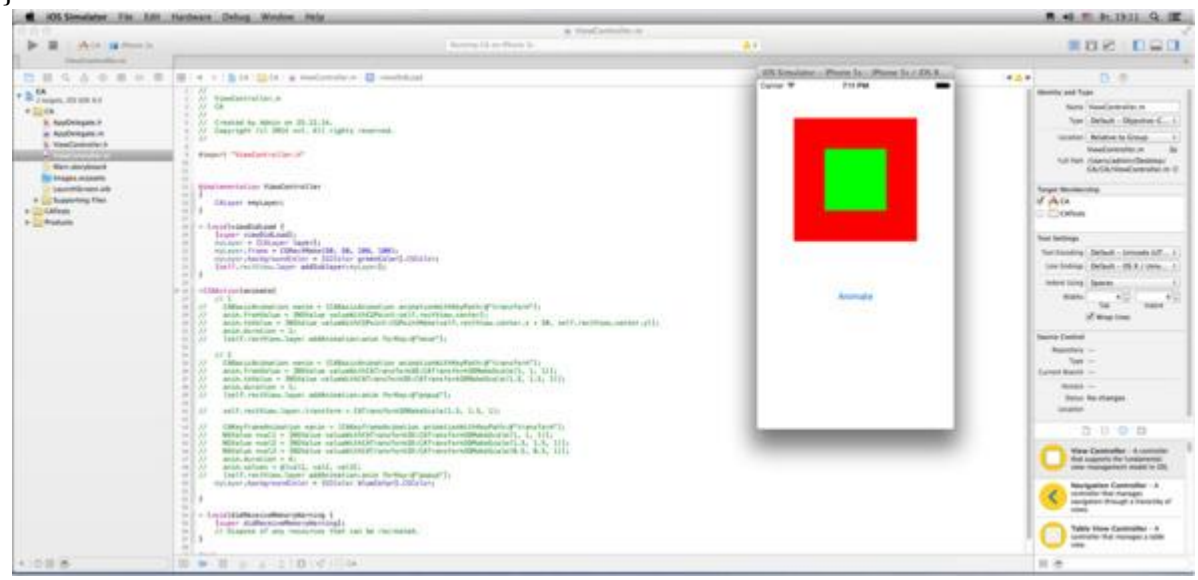
Посмотрим, как это работает:



Неявная анимация

Создадим и добавим нашей rectView новый слой. Сразу зададим backgroundColor.

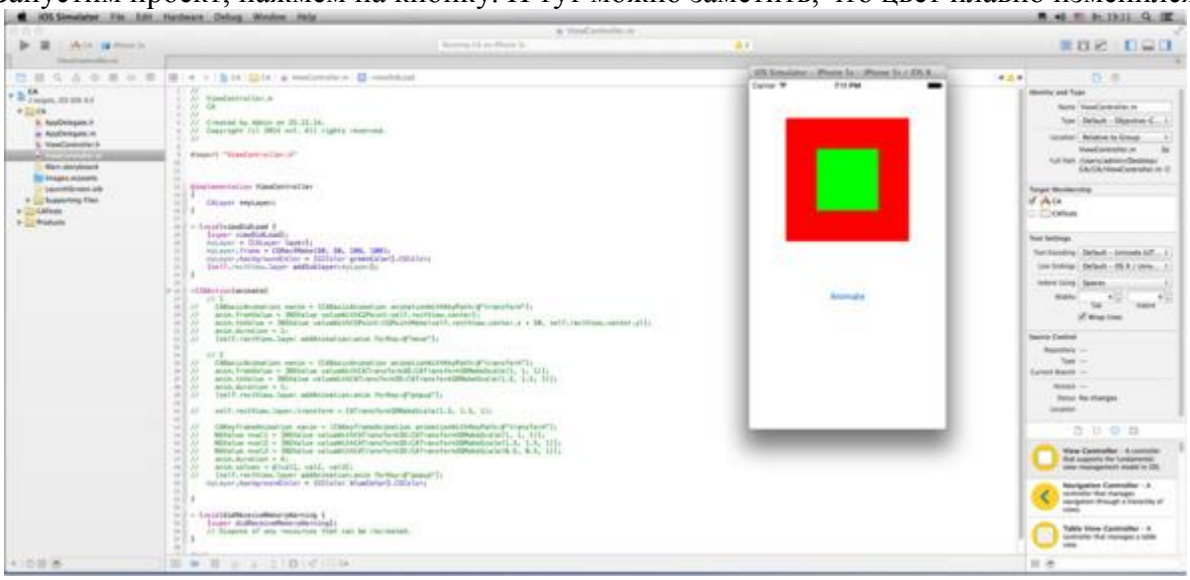
```
@implementation ViewController
{
CALayer *myLayer;
}

— (void) viewDidLoad {
[super viewDidLoad];
myLayer = [CALayer layer];
myLayer.frame = CGRectMake (50, 50, 100, 100);
myLayer.backgroundColor = [UIColor greenColor].CGColor;
[self.rectView.layer addSublayer: myLayer];
}
```

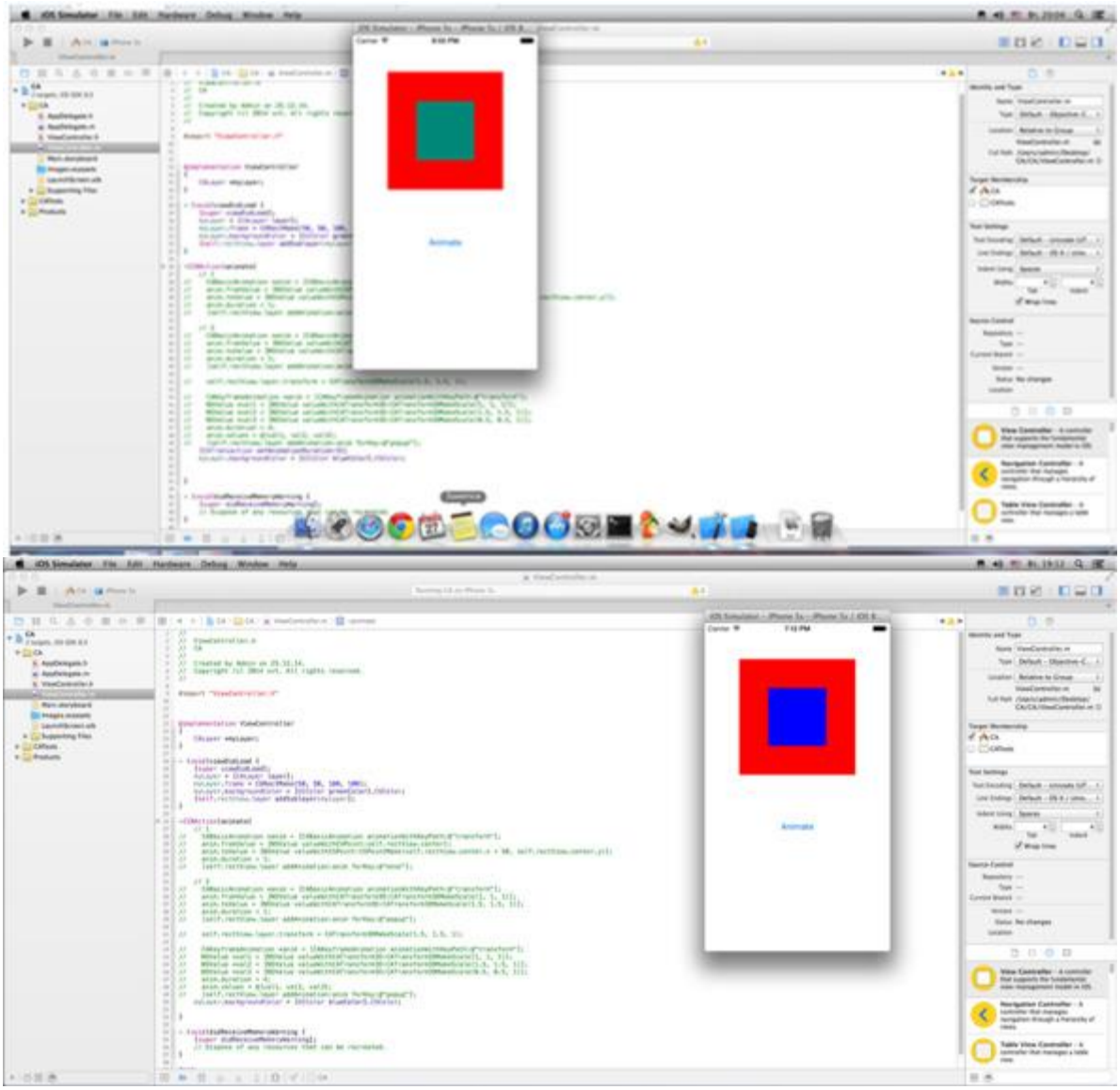


Теперь в методе animate добавим изменение цвета слоя.

```
— (IBAction) animate {
myLayer.backgroundColor = [UIColor blueColor].CGColor;
}
```

Запустим проект, нажмём на кнопку. И тут можно заметить, что цвет плавно изменился.





Но мы ведь не задавали никакой анимации, не так ли?

Дело в том, что при при изменении свойств самостоятельного слоя (не того, что принадлежит UIView!) анимация выполняется сама (создается CABasicAnimation, присваиваются начальные и конечные значения, анимация прикрепляется к слою и т.д.). При этом, при изменении сразу нескольких свойств, они группируются в одну так называемую транзакцию, которая выполняется каждый раз при выполнении кода. Параметры транзакции можно изменять, но необходимо это делать при первом изменении свойства слоя.

Установка длительности анимации в 5 секунд:
[CATransaction setAnimationDuration:5];

Установка completionBlock:
[CATransaction setCompletionBlock:^ {
NSLog (@«complete»);
}];

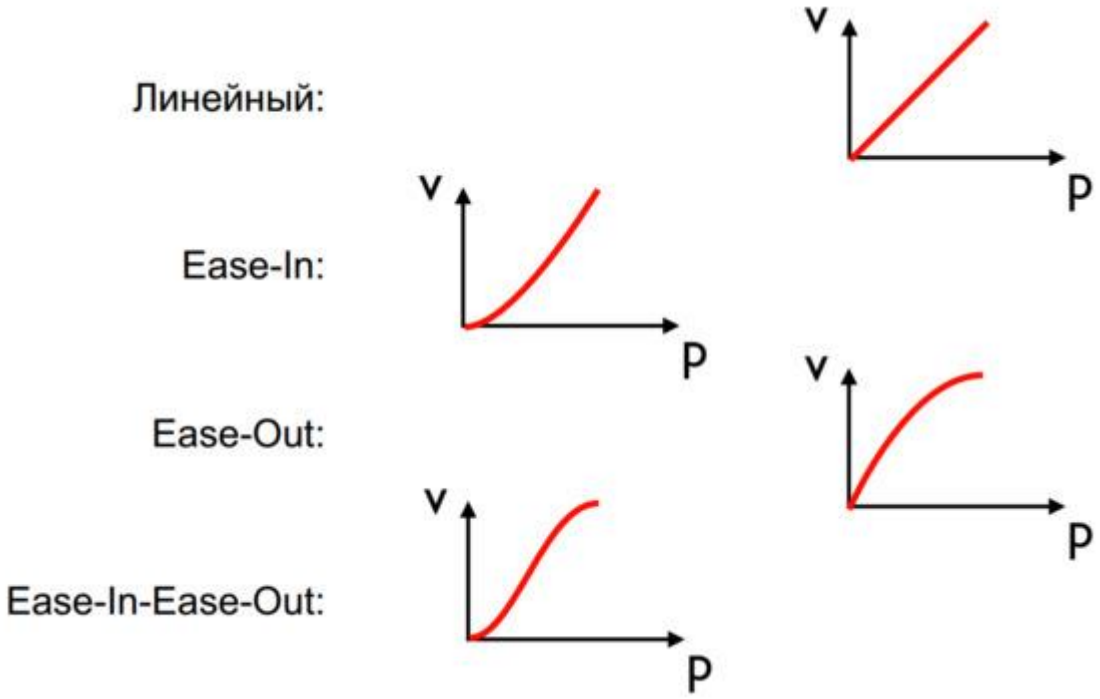
Установка timing — функции:
[CATransaction setAnimationTimingFunction: [CAMediaTimingFunction functionWithName: kCAMediaTimingFunctionEaseIn]];

Так же неявную анимацию можно отключить, если это нужно:
[CATransaction setDisableActions: YES];

Немного про временные функции

Временная функция — это функция, которая задают промежуточное значение свойства в определённый момент времени.

Эппл предоставляет 4 стандартных функции. Их графики можно увидеть ниже.



Создание собственных временных функций.

Для примера можно воспользоваться простой категорией, которая позволяет строить анимацию с собственной временной функцией, изменяемые свойства слоя должны быть типа float.

CAKeyframeAnimation+Parametric. h
typedef double (^KeyframeParametricBlock) (double time);

@interface CAKeyframeAnimation (Parametric)

+ (id) animationWithKeyPath: (NSString *) path
function: (KeyframeParametricBlock) block
fromValue: (double) fromValue
toValue: (double) toValue;
@end

CAKeyframeAnimation+Parametric. m
@implementation CAKeyframeAnimation (Parametric)

+ (id) animationWithKeyPath: (NSString *) path
function: (KeyframeParametricBlock) block
fromValue: (double) fromValue
toValue: (double) toValue {

CAKeyframeAnimation *animation =

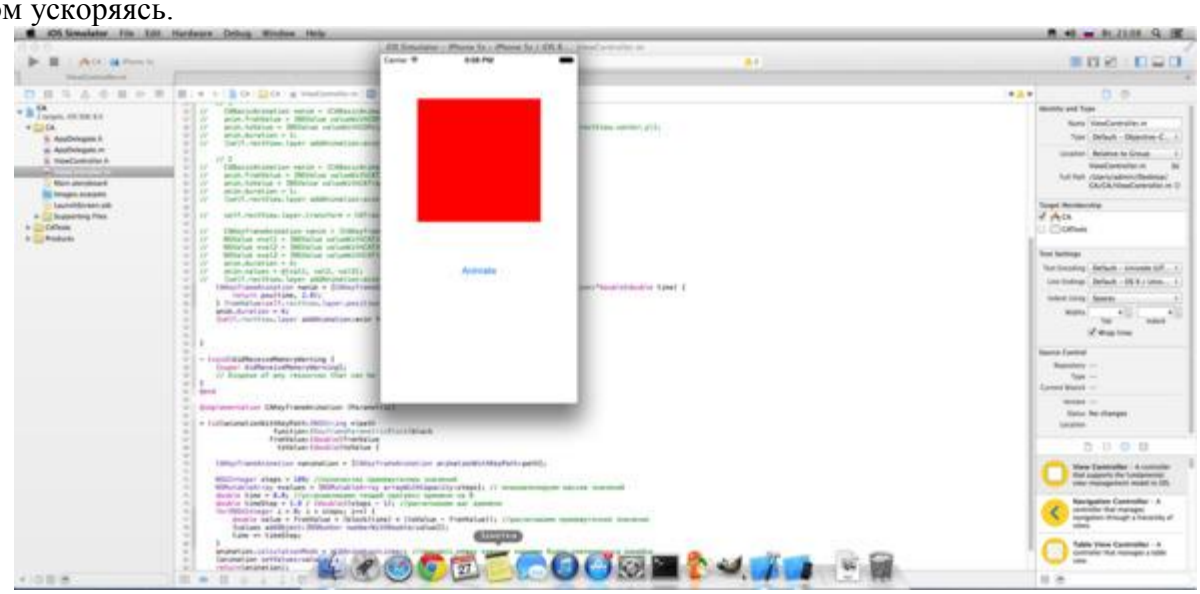
[CAKeyframeAnimation animationWithKeyPath: path];

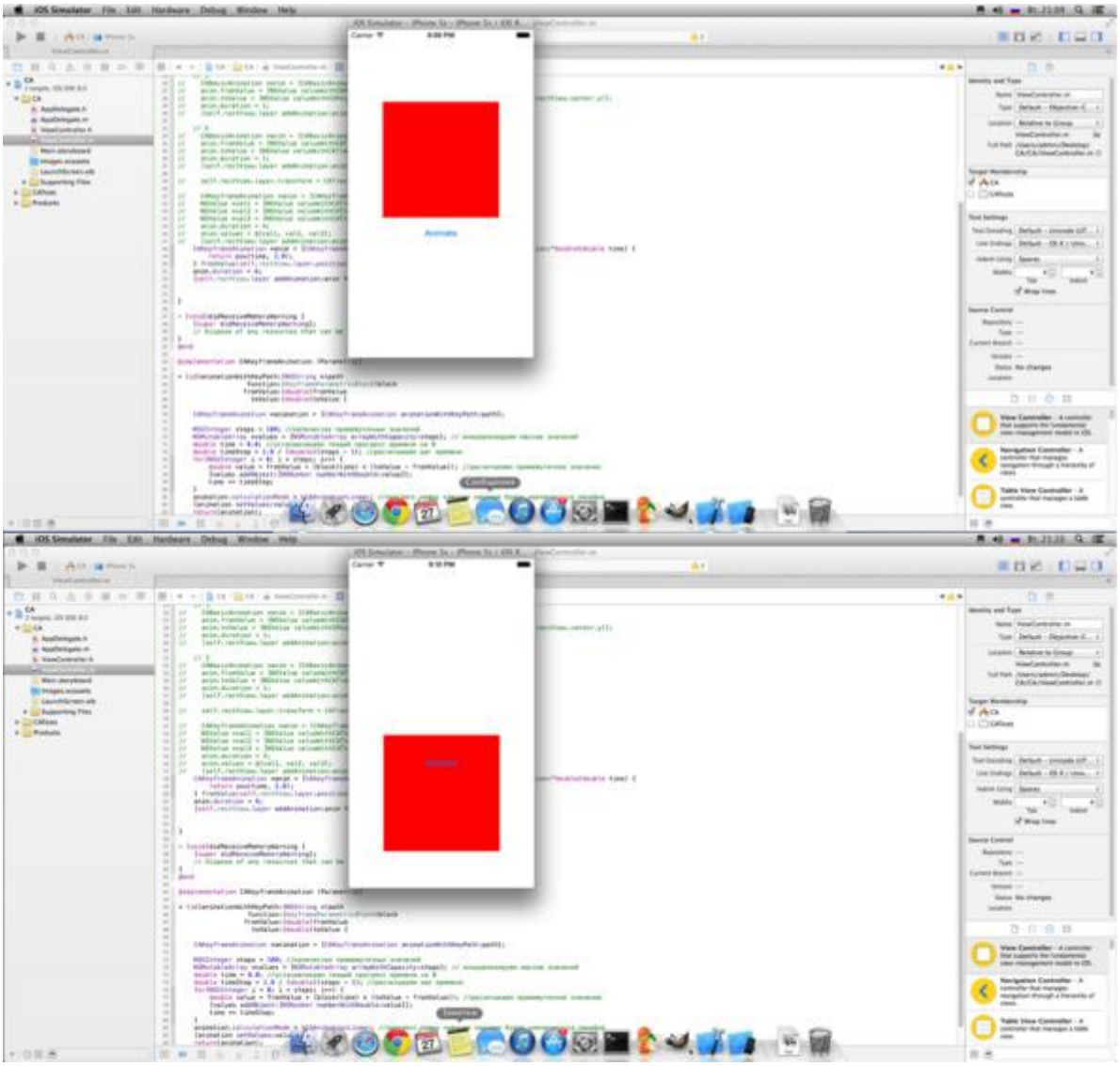
NSUInteger steps = 100; //количество промежуточных значений
NSMutableArray *values = [NSMutableArray arrayWithCapacity: steps]; // инициализируем массив значений
double time = 0.0; //устанавливаем текщий прогресс времени на 0
double timeStep = 1.0 / (double) (steps — 1); //расчитываем шаг времени
for (NSUInteger i = 0; i <steps; i++) {
double value = fromValue + (block (time) * (toValue — fromValue)); //расчитываем промежуточное значение
[values addObject: [NSNumber numberWithDouble: value]];
time += timeStep;
}
animation.calculationMode = kCAAnimationLinear; //прогресс между каждыми кадрами будет увеличиватся линейно
[animation setValues: values];
return (animation);
}
@end

Использование:

CAKeyframeAnimation *anim = [CAKeyframeAnimation animationWithKeyPath:@«position. y» function:^double (double time) {
return pow (time, 3.0); //задаём функцию, p=t^3, где p — прогресс, a t — время
} fromValue:self.rectView.layer. position. y toValue:600];
anim. duration = 6;
[self.rectView.layer addAnimation: anim forKey:@«key»];

Запустим приложение. Можно заметить, что UIView падает вниз, вначале медленно, потом ускоряясь.





Контроль анимации

Часто необходимо контролировать анимацию (остановить, поставить на паузу и т.д.).
Рассмотрим методы контроля завершения анимации.
Детектировать завершение анимации можно двумя способами — через блоки и делегата.

Через блоки

Тут придётся использовать CATransaction. Анимируем последним рассмотренным способом, в конце анимации скроем слой:

```
[CATransaction begin];
CAKeyframeAnimation *anim = [CAKeyframeAnimation animationWithKeyPath:@«position. y» function:^double (double time) {
return pow (time, 3.0);
} fromValue:self.rectView.layer. position. y toValue:600];
anim. duration = 5;
[CATransaction setCompletionBlock:^ {
self.rectView.layer. hidden = YES;
}];
[self.rectView.layer addAnimation: anim forKey:@«key»];
[CATransaction commit];
```

Здесь всё просто: мы создаем новую транзакцию, в ней создаём анимацию и прикрепляем её к слою. Назначаем транзакции completionBlock:^

{
self.rectView.layer. hidden = YES
}

Затем мы закрываем транзакцию

Через делегата

Объявляем себя делегатом:

CAKeyframeAnimation *anim = [CAKeyframeAnimation animationWithKeyPath:@«position. y» function:^double (double time) {
return pow (time, 3.0);
} fromValue:self.rectView.layer. position. y toValue:600];
anim. duration = 5;
anim.delegate = self;
[self.rectView.layer addAnimation: anim forKey:@«key»];

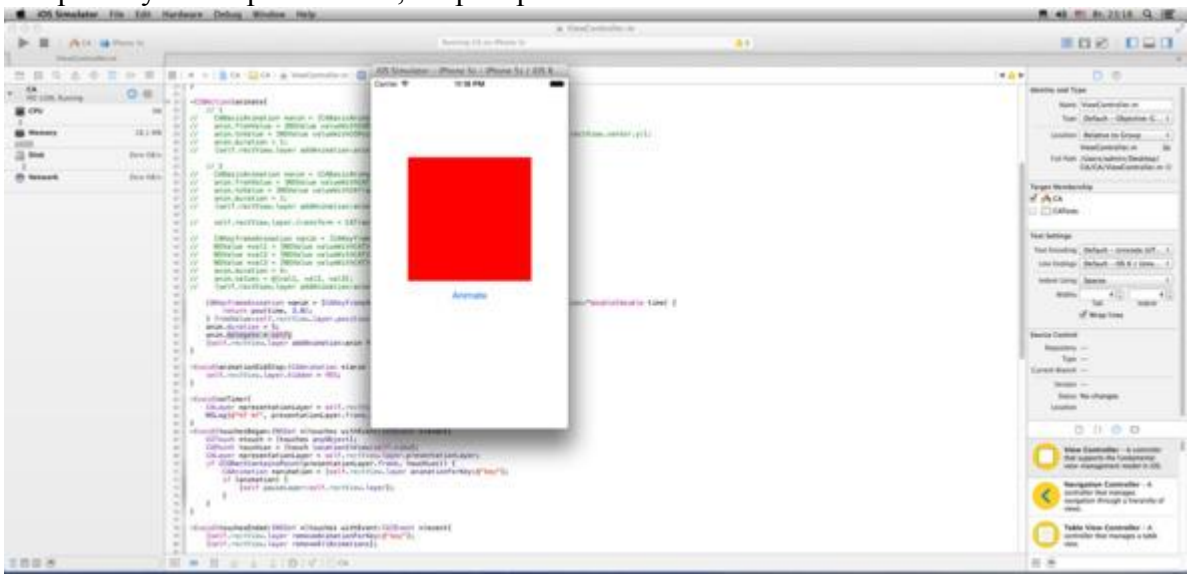
Реализовываем метод animationDidStop: finished:

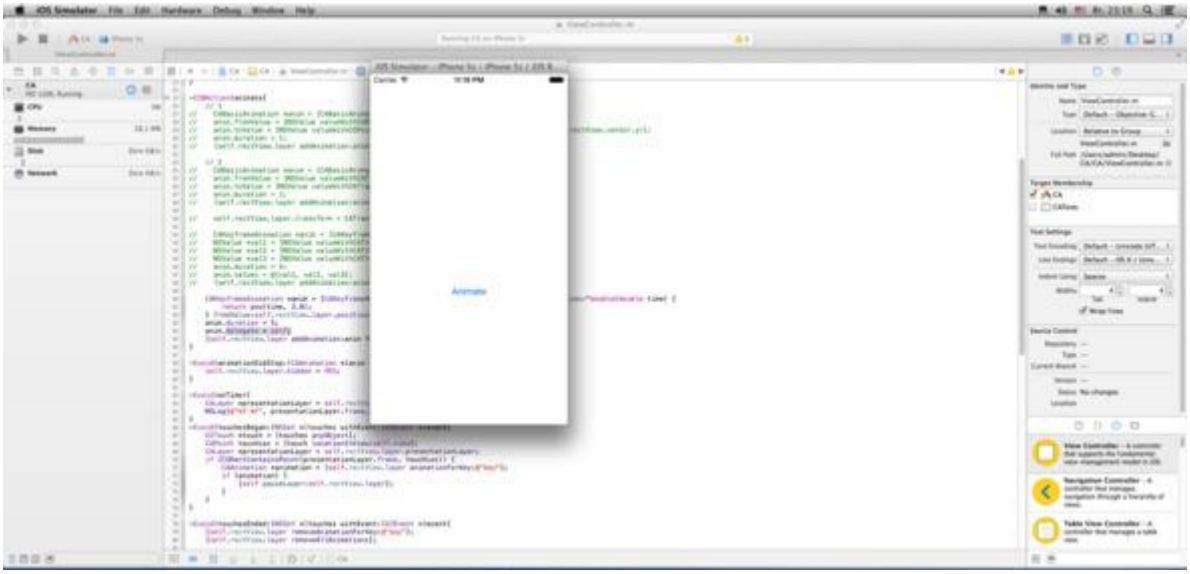
— (void) animationDidStop: (CAAnimation *) anim finished: (BOOL) flag {
self.rectView.layer. hidden = YES;
}

В котором скрываем слой.

В случае, если анимация была завершена, в качестве флага будет передано YES. Если анимация была приостановлена (об этом далее), то будет передано NO.

Теперь запустим приложение, и проверим.





Для паузы/возобновления анимации слоя Apple рекомендует реализовывать следующие методы:

```
— (void) pauseLayer: (CALayer*) layer {
CFTimeInterval pausedTime = [layer convertTime: CACurrentMediaTime () fromLayer: nil];
layer. speed = 0.0;
layer. timeOffset = pausedTime;
}

— (void) resumeLayer: (CALayer*) layer {
CFTimeInterval pausedTime = [layer timeOffset];
layer. speed = 1.0;
layer. timeOffset = 0.0;
layer.beginTime = 0.0;
CFTimeInterval timeSincePause = [layer convertTime: CACurrentMediaTime () fromLayer: nil] — pausedTime;
layer.beginTime = timeSincePause;
}
```

Допустим, мы хотим при тапе на движущийся UIView остановить его движение до тех пор, пока не отпустим палец.

Сделаем это следующим образом:

```
— (void) touchesBegan: (NSSet *) touches withEvent: (UIEvent *) event {
UITouch *touch = [touches anyObject];
CGPoint touchLoc = [touch locationInView:self.view];
CALayer *presentationLayer = self.rectView.layer.presentationLayer;
if (CGRectContainsPoint(presentationLayer.frame, touchLoc)) {
CAAnimation *animation = [self.rectView.layer animationForKey:@«key»];
if (animation) {
[self pauseLayer:self.rectView.layer];
}
}
}
```

Разберёмся, что здесь происходит. Получаем координаты касания:

```
UITouch *touch = [touches anyObject];
CGPoint touchLoc = [touch locationInView:self.view];

Получаем видимый слой
CALayer *presentationLayer = self.rectView.layer.presentationLayer;

Далее проверяем, принадлежит ли касание фрейму этого слоя.
if (CGRectContainsPoint(presentationLayer.frame, touchLoc)) {

}
```

Почему именно фрейму слоя, а не фрейму UIView? На самом деле, при анимации фрейм UIView не изменяется! Изменяется фрейм видимого слоя.

Осталось «найти» среди всех анимаций, что есть у слоя, ту, которая нужна нам (вот зачем нужен Key). Обязательно проверяем на nil, потому что после завершения анимация удаляется со слоя.

```
CAAnimation *animation = [self.rectView.layer animationForKey:@«key»];
if (animation) {
[self pauseLayer:self.rectView.layer];
}
```

Стоит отметить, в случае присваивания себя делегатом, при паузе анимации вызовется метод animationDidStop: finished:, при этом в качестве параметра finished, будет передано NO.

Аналогичным образом реализуем возобновление анимации.

```
— (void) touchesEnded: (NSSet *) touches withEvent: (UIEvent *) event {
UITouch *touch = [touches anyObject];
CGPoint touchLoc = [touch locationInView:self.view];
CALayer *presentationLayer = self.rectView.layer.presentationLayer;
if (CGRectContainsPoint(presentationLayer.frame, touchLoc)) {
CAAnimation *animation = [self.rectView.layer animationForKey:@«key»];
if (animation) {
[self resumeLayer:self.rectView.layer];
}
}
}
```

Для удаления конкретной анимации нужно воспользоваться методом removeAnimationForKey:

```
[self.rectView.layer removeAnimationForKey:@«key»];
```

Для удаления всех анимаций — методом removeAllAnimations:

```
[self.rectView.layer removeAllAnimations];
```

GCD

GCD или **Grand Central Dispatch** — механизм распараллеливания задач, представленный в iOS 4 и Mac OS X 10.6. Суть механизма в том, что реализация многопоточности скрывается от программиста. Всю «заботу» о создании потоков берет на себя GCD. Утверждается, что задачи GCD легковесны и требуют меньше процессорного времени, чем создание потоков. Получается, что все что требуется от программиста — определить какие задачи выполнять и поставить в нужную очередь, а GCD уже разберется со всем остальным.

Выполнение кода в фоне

Для этого есть функция dispatch_async.

dispatch_async (dispatch_get_global_queue (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^ {
// Код, который должен выполниться в фоне
});

Эта функция принимает в качестве аргумента очередь, в которой будет исполняться блок. Всего их 5: 4 фоновых, и одна главная. В примере в фунцкцию dispatch_get_global_queue мы передаем приоритет очереди, которую хотим получить. Вот они:

DISPATCH_QUEUE_PRIORITY_HIGH
DISPATCH_QUEUE_PRIORITY_DEFAULT
DISPATCH_QUEUE_PRIORITY_LOW
DISPATCH_QUEUE_PRIORITY_BACKGROUND

Все описанные выше очереди это так называемые Concurrent-очереди. Задачи в них выполняются параллельно. В таких очередях GCD сам управляет потоками и создает нужно количество потоков для выполнения задач. Второй аргумент функции dispatch_get_global_queue всегда ноль (он зарезервирован на будущее). Для получения главной очереди, необходимо использовать функцию dispatch_get_main_queue ()

Кроме concurrent-очередей существуют еще и Serial-очереди, задачи в которых выполняются последовательно. Для каждой serial-очереди создается отдельный поток. Serial-очередь выполняет только одну задачу. После того, как очередная задача выполнена — берется следующая, поставленная в очередь на выполнение. Эта очередь работает по принципу FIFO (First in — First out, «Первый пришел — первый ушел»).

Serial-очередь создается вот так:

dispatch_queue_t serialQueue = dispatch_queue_create("com.myapp. queue», NULL)

Когда очередь вам больше не нужна, для нее следует вызвать функцию dispatch_release.

dispatch_release (serialQueue);

Это же касается и Concurrent-очередей.

Синхронное выполнение кода

Тут нам поможет функция dispatch_sync

dispatch_sync (dispatch_get_main_queue (), ^ {
/* Код, который нужно выполнить в главном потоке. При этом следующий за ним код не будет выполнятся до его завершения */})

Выполнение кода один раз

```
static dispatch_once_t onceToken;
dispatch_once (&onceToken, ^ {
//Код, который будет выполнен один раз
});
```

Apple гарантирует, что блок кода будет выполнен один раз. Функция dispatch_once активно используется при реализации паттерна Singleton. В качестве аргумента принимает блок, который нужно выполнить и onceToken. Что такое onceToken? Это, можно сказать, уникальный ид выполняемой задачи. Ему можно не присваивать значение (тогда оно случайным образом присвоится компилятором), но стоит следить, чтобы 2 разные задачи имели разные токены.

Например, блок кода из второй функции не будет выполнен, так как уже с таким токеном был выполнен первый.

```
— (void) viewDidLoad {
[super viewDidLoad];
static dispatch_once_t onceToken;
dispatch_once (&onceToken, ^ {
NSLog (@«block1»);
});

dispatch_once (&onceToken, ^ {
NSLog (@«block2»);
});
}
```

Для того, чтобы этого не произошло, используем 2 разных токена.

```
— (void) viewDidLoad {
static dispatch_once_t onceToken1;
dispatch_once (&onceToken1, ^ {
NSLog (@«block1»);
});

static dispatch_once_t onceToken2;
dispatch_once (&onceToken2, ^ {
NSLog (@«block2»);
});
}
```

Выполнение кода через определённое время

```
double delayInSeconds = 3;
dispatch_after (dispatch_time (DISPATCH_TIME_NOW, (int64_t) (delayInSeconds * NSEC_PER_SEC)), dispatch_get_main_queue (), ^ {
//Код
});
```

Обратите внимание на то, что нет гарантий, что код запуститься сразу по истечении времени. Когда пройдет время, задача станет в очередь на выполнение, а начнет она выполняться или нет — зависит уже от нагруженности самой очереди. Но, обычно, задачи стартуют во время или с небольшой погрешностью.

Время, когда запускать задачу на выполнение представлено структурой dispatch_time_t, которая создается функцией dispatch_time**.** Она определяет сколько времени (второй аргумент), прошло от конкретного времени (первый аргумент). В данном случае мы используем текуще время DISPATCH_TIME_NOW и отсчитываем от него три секунды. Время отсчета задается в нано секундах. Если мы хотим использовать время в секундах, то нам нужно умножать количество секунд на NSEC_PER_SEC (наносекунд в секунде). Если мы хотим отсчитать милисикунды, можно использовать NSEC_PER_MSEC).

Выполнение кода несколько раз.

Для этого необходимо воспользоваться функцией dispatch_apply

```
dispatch_queue_t queue =
dispatch_get_global_queue (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch_apply (10, queue, ^ (size_t index) {
// блок выполнится 10 раз
// в блок передается номер запуска
});
```

dispatch_apply распаралеливает выполнение задач. Например, если запустить вот такой код:

```
dispatch_apply (10, queue, ^ (size_t index) {
NSLog (@"%@, %zu», [NSDate date], index);
});
```

В консоли мы можем увидеть вот такое:

2015-01-28 01:13:08.334 gcd [856:20040] 2015-01-27 15:13:08 +0000, 2
2015-01-28 01:13:08.340 gcd [856:20040] 2015-01-27 15:13:08 +0000, 4
2015-01-28 01:13:08.340 gcd [856:20040] 2015-01-27 15:13:08 +0000, 5
2015-01-28 01:13:08.340 gcd [856:20040] 2015-01-27 15:13:08 +0000, 6
2015-01-28 01:13:08.340 gcd [856:20040] 2015-01-27 15:13:08 +0000, 7
2015-01-28 01:13:08.340 gcd [856:20040] 2015-01-27 15:13:08 +0000, 8
2015-01-28 01:13:08.334 gcd [856:19999] 2015-01-27 15:13:08 +0000, 0
2015-01-28 01:13:08.339 gcd [856:20039] 2015-01-27 15:13:08 +0000, 1
2015-01-28 01:13:08.340 gcd [856:20040] 2015-01-27 15:13:08 +0000, 9
2015-01-28 01:13:08.339 gcd [856:20041] 2015-01-27 15:13:08 +0000, 3

Остановка/восстановление очереди

Создадим очередь, и остановим её.

```
dispatch_queue_t queue = dispatch_queue_create("com.myapp.myqueue», DISPATCH_QUEUE_CONCURRENT);
dispatch_suspend (queue);
```

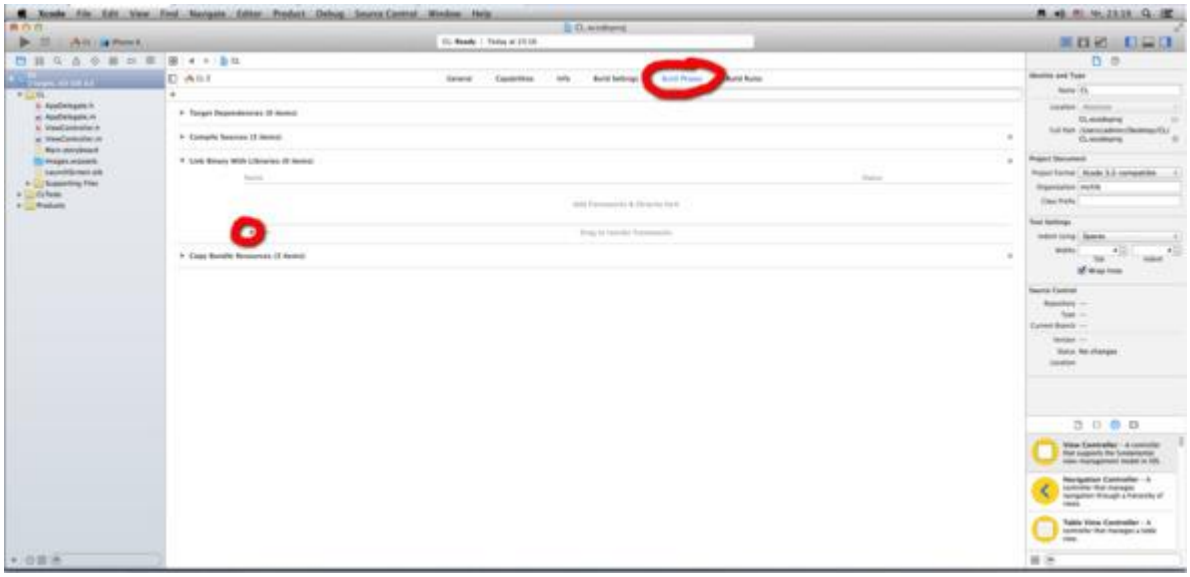
При этом, очередь не будет выполнять новые задачи, а те, которые выполнялись, продолжат выполняться. Для того чтобы восстановить очередь, используем функцию dispatch_resume

```
dispatch_resume (queue);
```

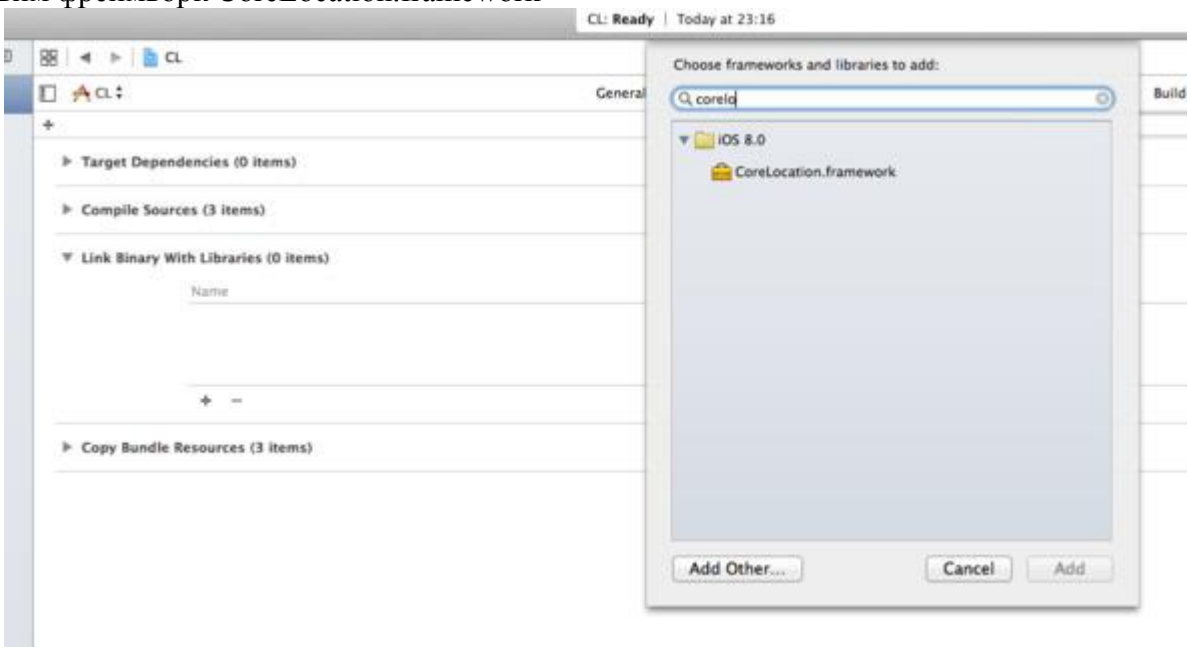
Основы CoreLocation

Для начала создадим новый проект, выберем шаблон Single View Application.

Откроем свойства проекта, выберем во вкладке Build Phases секцию Link binary with libraries



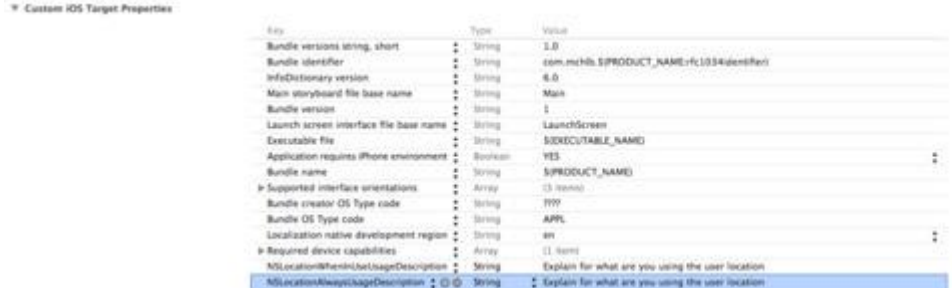
Добавим фреймворк CoreLocation.framework



Во вкладке Info в секции Custom iOS target properties добавим 2 новых пары ключ-значение:

NSLocationWhenInUseUsageDescription (string): Explain for what are you using the user location

NSLocationAlwaysUsageDescription (string): Explain for what are you using the user location



Импортируем заглавный header в ViewController. h

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

А так же примем протокол CLLocationManagerDelegate

@interface ViewController: UIViewController <CLLocationManagerDelegate>

Объявим CLLocationManager

```
@implementation ViewController
{
CLLocationManager *manager;
}
```

Теперь в методе viewDidLoad проинициализируем locationmanager, установим себя делегатом и назначим точность. На выбор нам предоставлены следующие значения желаемой точности получаемых координат

```
extern const CLLocationAccuracy kCLLocationAccuracyBestForNavigation
extern const CLLocationAccuracy kCLLocationAccuracyBest;
extern const CLLocationAccuracy kCLLocationAccuracyNearestTenMeters;
extern const CLLocationAccuracy kCLLocationAccuracyHundredMeters;
extern const CLLocationAccuracy kCLLocationAccuracyKilometer;
extern const CLLocationAccuracy kCLLocationAccuracyThreeKilometers;

— (void) viewDidLoad {
[super viewDidLoad];

manager = [CLLocationManager new];
manager.delegate = self;
manager.desiredAccuracy = kCLLocationAccuracyBest;
}
```

Теперь необходимо уведомить пользователя, желает ли он разрешить приложению использовать своё местонахождение (для этого мы также добавляли новые пары ключ/значение в Info). Эта процедура является обязательной для получения координат пользователя.

```
if ([manager respondsToSelector:@selector (requestWhenInUseAuthorization)])
{
[manager requestWhenInUseAuthorization];
}
```

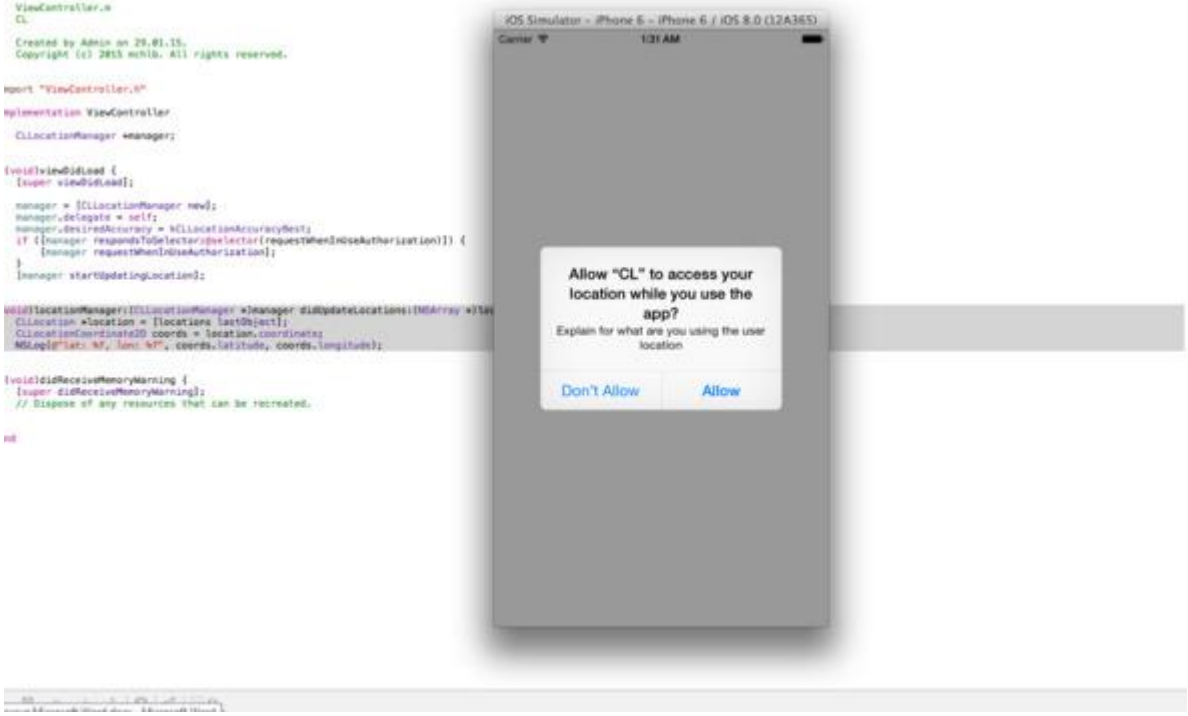
Всё, осталось только запустить обновление местоположения и можно реализовывать методы делегата CLLocationManager

```
[manager startUpdatingLocation]
```

Реализуем метод — (void) locationManager: (CLLocationManager *) manager didUpdateLocations: (NSArray *) locations, который будет вызываться всякий раз, когда приложение будет получать координаты.

```
— (void) locationManager: (CLLocationManager *) manager didUpdateLocations: (NSArray *) locations {
CLLocation *location = [locations lastObject];
```

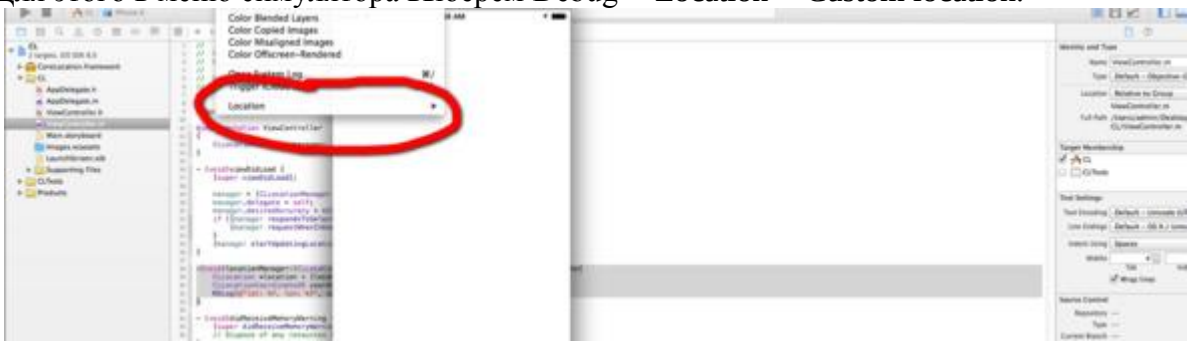
CLLocationCoordinate2D coords = location.coordinate;
NSLog (@«lat: %f, lon: %f», coords.latitude, coords. longitude);
}
Попробуем запустить наше приложение.



Приложение запрашивает разрешение на использование координат пользователя. Нажмём «Allow».

Если вы используете реальное устройство, то в лог отобразятся координаты
2015-01-30 01:34:57.491 CL [1216:41521] lat: 54.000000, lon: 37.000000
2015-01-30 01:34:58.492 CL [1216:41521] lat: 54.000000, lon: 37.000000
2015-01-30 01:34:59.493 CL [1216:41521] lat: 54.000000, lon: 37.000000
2015-01-30 01:35:00.494 CL [1216:41521] lat: 54.000000, lon: 37.000000
2015-01-30 01:35:01.496 CL [1216:41521] lat: 54.000000, lon: 37.000000
2015-01-30 01:35:02.496 CL [1216:41521] lat: 54.000000, lon: 37.000000

В случае использования симулятора необходимо симулировать свое местоположение. Для этого в меню симулятора выберем Debug-> Location-> Custom location.



Social framework

Иногда в приложении необходимо реализовать функцию «Поделится в соц сетях». Для социальных сетей Facebook и Twitter существует Social framework.

Для начала создадим новый проект, выберем шаблон Single View Application. Откроем ViewController. m

Импортируем заглавный хедер

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

Добавим кнопку «Поделиться» на экран.

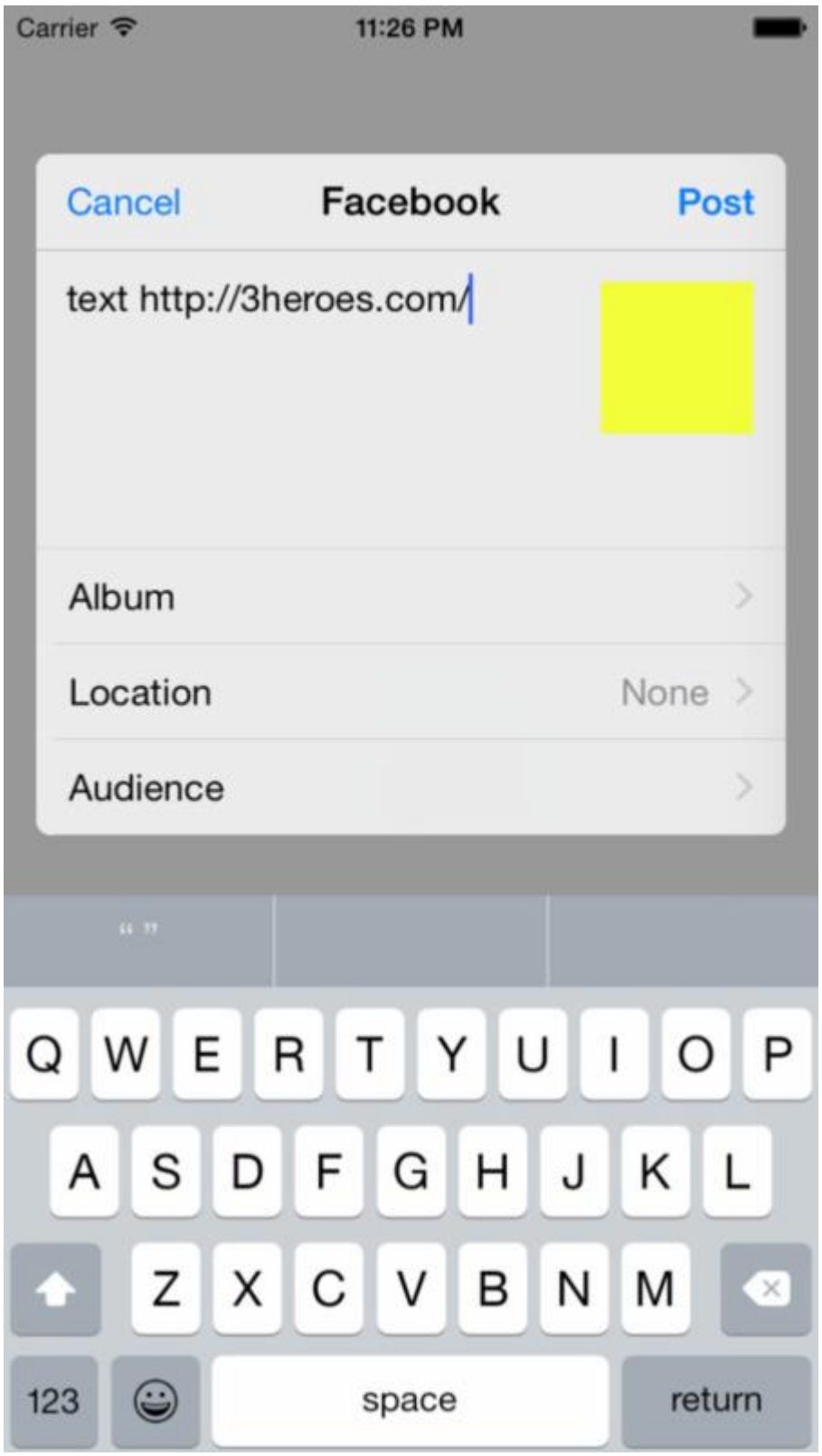
```
— (void) viewDidLoad {
[super viewDidLoad];
UIButton *button = [UIButton buttonWithType: UIButtonTypeSystem];
button. center = self.view. center;
[button setTitle:@«Share» forState: UIControlStateNormal];
[button sizeToFit];
[button addTarget: self action:@selector (share) forControlEvents: UIControlEventTouchUpInside];
[self.view addSubview: button];
}
```

Реализуем метод share, который будет вызыватся при нажатии кнопки.

```
— (void) share {
SLComposeViewController *share = [SLComposeViewController composeViewControllerForServiceType: SLServiceTypeFacebook]; //инициализируем «шарильщик» с типом «Facebook»
[share setInitialText:@«text»]; // устанавливаем начальный текст
[share addURL: [NSURL URLWithString:@"http://3heroes.com/"]]; //ссылку
[share addImage: [UIImage imageNamed:@"image.png»]]; // изображение
[self presentViewController: share animated: YES completion: nil]; // открываем окно «шарильщика»
}
```

Для Twitter необходимо установить тип SLServiceTypeTwitter.

Запустим приложение и нажмём на кнопку.



Как можно заметить, данный фреймворк очень лёгок в использовании.

NSURLConnection

Для начала создадим новый проект, выберем шаблон Single View Application. Откроем ViewController. m

В первую очередь для работы с NSURLConnection необходимо принять протокол NSURLConnectionDelegate и NSURLConnectionDataDelegate.

@interface ViewController () <NSURLConnectionDelegate, NSURLConnectionDataDelegate>

@end

Обьявим переменую типа NSMutableData — в неё мы будем загружать данные

```
@implementation ViewController
{
NSMutableData *recievedData;
}
```

Подготовим POST запрос. Для этого добавим в метод ViewDidLoad следущий код:

```
NSURL *url = [NSURL URLWithString:@"http://api.jankypost.com/api"]; // URL запроса

NSString *parameters = @«one=SomeParametr1&two=SomeParameter2»; // Параметры запроса

NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL: url]; //создаем запрос
request. timeoutInterval = 15.0f; //устанавливаем время попытки подключения
request.HTTPMethod = @«post»; // устанавливаем метод запроса (post/get)
request.HTTPBody = [parameters dataUsingEncoding: NSUTF8StringEncoding]; //добавляем параметры в тело запроса
```

Для GET запроса HTTPBody устанавливать не нужно, так как параметры указываются прямо в URL и разделяются от основного URL знаком»?». При этом HTTPMethod указывать необязательно. Например

```
NSString *parameters = @«one=SomeParametr1&two=SomeParameter2»;

NSURL *url = [NSURL URLWithString: [NSString stringWithFormat:@"http://api.jankypost.com/api?%@», parameters]];
NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL: url]; request. timeoutInterval = 15.0f;
```

Замечание: если строка параметров содержит пробелы, то её необходимо преобразовать в escape-последовательность:

```
parameters = [parameters stringByAddingPercentEscapesUsingEncoding: NSUTF8StringEncoding];
```

Теперь необходимо инициализировать подключение. Обязательно устанавливаем себя делегатом NSURLConnection.

```
NSURLConnection *connection = [[NSURLConnection alloc] initWithRequest: request delegate: self];
if (connection) {
recievedData = [NSMutableData new];
NSLog (@«Init Success»); //если инициализация произошла успешно
} else {
NSLog (@«Init Error!»); // если при инициализации произошла ошибка
```

}

Теперь необходимо реализовать методы делегата NSURLConnectionDelegate и NSURLConnectionDataDelegate :

— (void) connection: (NSURLConnection *) connection didReceiveResponse: (NSURLResponse *) response

Этот метод вызывается при первом получении ответа от сервера, это сигнализирует об успешном соединении

— (void) connection: (NSURLConnection *) connection didReceiveData: (NSData *) data

Этот метод вызывается при получении новых данных

— (void) connectionDidFinishLoading: (NSURLConnection *) connection

Этот метод вызывается при завершении загрузки

— (void) connection: (NSURLConnection *) connection didFailWithError: (NSError *) error

Этот метод вызывается при ошибке соединения

Реализация методов:

```
— (void) connection: (NSURLConnection *) connection didReceiveResponse: (NSURLResponse *) response {
[recievedData setLength:0];
}

— (void) connection: (NSURLConnection *) connection didReceiveData: (NSData *) data {
[recievedData appendData: data];
}

— (void) connectionDidFinishLoading: (NSURLConnection *) connection {
NSLog (@«Finish»);
}

— (void) connection: (NSURLConnection *) connection didFailWithError: (NSError *) error {
recievedData = nil; //освобождаем данные
NSLog (@«Error»);
}
```

Посмотрим, что же представляет из себя recievedData после окончания загрузки.

Итак, данные получены, но они — не более чем ничего не несущая в себе последовательность байт, и их необходимо распарсить.

Запрос по адресу http://api.jankypost.com/api возвращает json (наиболее популярный на сегодняшний день формат представления данных) словарь типа: {

«параметр1 запроса»: «значение»

.

.«параметрN запроса»: «значение»

«тип запроса»: «post/get»

«server_time»: «значение»

}

Для того, чтобы распарсить полученные данные, воспользуемся классом NSJSONSerialization.

Добавим следующий код в метод connectionDidFinishLoading:

```
NSError *error;
NSDictionary *parsedData = [NSJSONSerialization JSONObjectWithData: recievedData options: NSJSONReadingMutableContainers error:&error];
if (!error) {
NSLog (@"%@», parsedData.description);
} else {
NSLog (@«Parsing error!»);
}
```

Обратим внимание, что в качестве аргумента error мы присваиваем адрес экземпляра NSError.

В случае успешного парсинга, экземпляр NSError будет Nil.

Посмотрим, что из себя представляет parsedData. Консоль выведет следующее

{

method = POST;

one = SomeParametr1;

«server_time» = «1415626204.51»;

two = SomParameter2;

}

Так же, стоит обратить внимание, что на один и тот же запрос сервер может возвращать как JSON словарь так и JSON массив (например, в случае ошибки на стороне сервера возвращать словарь, а в случае успеха — массив), и если распарсить не в тот тип данных (например словарь в массив), и после этого обратиться к нему, то приложение выдаст

исключение. Для того, чтобы этого не произошло, необходимо парсить в тип id, а потом уже проверять на принадлежность к тому или иному классу.

```
id parsedData = [NSJSONSerialization JSONObjectWithData: recievedData options: NSJSONReadingMutableContainers error:&error];
if ([parsedData isKindOfClass: [NSDictionary class]]) {
//действие для словаря
} else if ([parsedData isKindOfClass: [NSArray class]]) {
//действие для массива
}
```

Заключение

Использование инструментария и аппаратных средств в совокупности для тестирования и отладки приложений одновременно представляется интересным подходом, который как видится автору будет очень востребован в будущем. Данная идеология может использоваться и на станках с ЧПУ и при подготовки специалистов из различных областей.

Данная работа не претендует на всеобъемлющий труд и попытку описать все возможности и «подводные камни», которые могут встретиться на пути начинающего разработчика мобильных приложений под платформу компании Apple. Автор попытался зафиксировать моменты с которыми каждому придется столкнуться и «сгладить углы».

В глобальной сети интернет существует множество форумов, книг, конференций, обсуждений связанных с разработкой IOS приложений и порой какие-то моменты трудно сразу найти.

Надеюсь эта книга окажется полезной.

Список использованных источников

1. Аллан, А. Программирование для мобильных устройств на iOS: Профессиональная разработка приложений для iPhone, iPad, and iPod Touch / А. Аллан.. — СПб.: Питер, 2013. — 416 с.

2. Васильев, А. Е. Микроконтроллеры. Разработка встраиваемых приложений / А. Е. Васильев. — СПб.: BHV, 2008. — 304 с.

3. Веллинг, Л. Разработка Web-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. — М.: Вильямс, 2013. — 848 с.

4. Гарнаев, А. Мастер Visual Basic.NET. Разработка приложений / А. Гарнаев. — СПб.: BHV, 2002. — 624 с.

5. Голощапов А. Л. Google Android: программирование для мобильных устройств. — СПб.:БХВ-Петербург, 2011

6. Гринберг, М. Разработка веб-приложений с использованием Flask на языке Python / М. Гринберг. — М.: ДМК, 2014. — 272 с.

7. Далримпл М., Кнастер С. Objective-C 2.0 и программирование для Mac. Учебник и примеры. Вильямс, 2009, 320 с.

8. Дари, К. AJAX и PHP. Разработка динамических веб-приложений / К. Дари, Б. Бринзаре, Ф. Черчез-Тоза, М. Бусика. — СПб.: Символ-плюс, 2015. — 336 с.

9. Джонсон, Г. Разработка клиентских веб-приложенийи на платформе. NET Framework: экзамен 70—528 / Г. Джонсон. — М.: Русская редакция, 2008. — 768 с.

10. Джордж Шеферд. Программирование на MicrosoftVisual C++.NET. 2010г.

11. Дэйв Марк, Джек Наттинг, ДжеффЛамарш; Переводчики: И. Берштейн, Дмитрий Клюшин, Игорь Красиков, Н. Ручко. Разработка приложений для iPhone, iPad и iPodtouch с использованием iOS SDK. 2011г.

12. Дэйв, М. iOS 5 SDK. Разработка приложений для iPhone, iPad и iPod touch / М. Дэйв, Н. Джек. — М.: Вильямс, 2012. — 672 с.

13. Есенин, С. А. DirectX и Delphi: разработка графических и мультимедийных приложений / С. А. Есенин. — СПб.: BHV, 2006. — 512 с.

14. Здзиарски, Д. iPhone SDK. Разработка приложений / Д. Здзиарски. — СПб.: BHV, 2013. — 512 с.

15. Здзиарски, Д. iPhone. Разработка приложений с открытым кодом / Д. Здзиарски. — СПб.: BHV, 2013. — 368 с.

16. Кирстен, В. Постреляционная СУБД Cache 5 Объектно-ориентированная разработка приложений / В. Кирстен, М. Иринger, Б. Реriг. — М.: Бином-Пресс, 2011. — 416 с.

17. Козловский, П. Разработка веб-приложений с использованием AngularJS / П. Козловский, П. Дарвин. — М.: ДМК, 2014. — 394 с.

18. Колисниченко, Д. Н. PHP и MySQL. Разработка Web-приложений / Д. Н. Колисниченко. — СПб.: БХВ-Петербург, 2013. — 560 с.

19. Колисниченко, Д. Н. Разработка Linux-приложений. / Д. Н. Колисниченко. — СПб.: BHV, 2012. — 432 с.

20. Крейг Хоккенбери; Переводчик: В. Порицкий. Разработка приложений под iPhone. Полное руководство. 2011г.

21. Кречмер, Р. Разработка приложений SAP R/3 на языке ABAP/4 / Р. Кречмер, В. Вейс. — М.: Лори, 2014. — 461 с.

22. Марк, Д. Разработка приложений для iPhone, iPad и iPod touch с использованием iOS SDK / Д. Марк, Д. Наттинг, Д. Ламарш. — М.: Вильямс, 2012. — 624 с.

23. Маркин, А. Разработка приложений баз данных в Delphi Самоучитель / А. Маркин. — М.: Диалог-МИФИ, 2013. — 160 с.

24. Масленникова О. Е., Попова И. В. Основы искусственного интеллекта Учебное пособие. Магнитогорск: МаГУ, 2008, 282 с.

25. Машнин, Т. С. Eclipse: разработка RCP-, Web-, Ajax- и Android-приложений / Т. С. Машнин. — СПб.: БХВ-Петербург, 2013. — 384 с.

26. Машнин, Т. С. Google App Engine Java и Google Web Toolkit: разработка Web-приложений / Т. С. Машнин. — СПб.: BHV, 2014. — 352 с.

27. Миковски, М. С. Разработка одностраничных веб-приложений / М. С. Миковски, Д. К. Пауэлл. — М.: ДМК, 2014. — 512 с.

28. Мэтт Нойбург. Программирование для iOS 7. ОсновыObjective-C, Xcode и Cocoa.2014г.

29. Нахавандипур, В. IOS. Разработка приложений для iPhone, iPad и iPod / В. Нахавандипур; Пер. с англ. О. Сивченко. — СПб.: Питер, 2013. — 864 с.

30. Нойбург М. Программирование для iOS 7. Основы Objective-C, Xcode и Cocoa. Вильямс, 2014, 384 с.

31. Нортроп, Т. Разработка защищенных приложений на Visual Basic. NET и Visual C#.NET: Учебный курс Microsoft / Т. Нортроп. — М.: Русская редакция, 2007. — 688 с.

32. Пахомов Б. С/С++ и MSVisual C++2008 для начинающих. 2009г.

33. Полещук, В. И. AutoCAD 2004: разработка приложений и адаптация. / В. И. Полещук. — СПб.: BHV, 2004. — 624 с.

34. Постолит, А. Visual Studio.NET: разработка приложений баз данных. / А. Постолит. — СПб.: BHV, 2003. — 544 с.

35. Прохоренок, Н. А. Python 3 и PyQt. Разработка приложений / Н. А. Прохоренок.. — СПб.: БХВ-Петербург, 2013. — 704 с.

36. Редькин, П. П. Прецизионные системы сбора данных семейства MSC12xx фирмы Texas Instruments: архитектура, программирование, разработка приложений. / П. П. Редькин. — М.: Додэка, 2006. — 608 с.

37. Руби, С. Гибкая разработка веб-приложений в среде Rails / С. Руби. — СПб.: Питер, 2013. — 464 с.

38. Рудикова, Л. В. Базы данных. Разработка приложений / Л. В. Рудикова. — СПб.: BHV, 2006. — 496 с.

39. Саммерфилд, М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++ / М. Саммерфилд. — М.: Символ-Плюс, 2011. — 560 с.

40. Саммерфилд, М. Программирование на языке Go Разработка приложений XXI века / М. Саммерфилд. — М.: ДМК Пресс, 2013. — 580 с.

41. Сафронов, М. Разработка веб-приложений в Yii 2 / М. Сафронов. — М.: ДМК, 2015. — 392 с.

42. Сергеенко, С. В. Разработка и проектирование Web-приложений Oracle Developer / С. В. Сергеенко. — М.: Интуит, 2014. — 456 с.

43. Соколова, Ю. С. Разработка приложений в среде Delphi. В 2 частях. Часть 2. Компоненты и их использование: Учебное пособие для вузов. / Ю. С. Соколова, С. Ю. Жулева. — М.: Горячая линия -Телеком, 2011. — 144 с.

44. Соколова, Ю. С. Разработка приложений в среде Delphi. Ч. 1. Общие приемы программирования., стер / Ю. С. Соколова, С. Ю. Жулева. — М.: ГЛТ, 2013. — 144 с.

45. Трещев И. А. Программирование для мобильных платформ. ФГБОУ ВО КнАГУ, 2018 — 100 с.

46. Уллман, Л Adobe AIR. Разработка приложений с помощью Ajax / Л Уллман. — СПб.: BHV, 2009. — 560 с.

47. Финкэнон, Д. Flash-реклама. Разработка микросайтов, рекламных игр и фирменных приложений с помощью Adobe Flash / Д. Финкэнон. — М.: Рид Групп, 2012. — 288 с.

48. Фиртман, М. jQuery Mobile: разработка приложений для смартфонов и планшетов / М. Фиртман; Пер. с англ. С. Иноземцев. — СПб.: БХВ-Петербург, 2013. — 256 с.

49. Форсье, Д. Django. Разработка веб-приложений на Python / Д. Форсье. — М.: Символ-Плюс, 2009. — 456 с.

50. Чан, У. Django. Разработка веб-приложений на Python / У. Чан, П. Биссекс, Д. Форсье. — СПб.: Символ-плюс, 2015. — 456 с.

51. Чедвик, Д. ASP.NET 4: разработка реальных веб-приложений с помощью ASP.NET MVC / Д. Чедвик. — М.: Вильямс, 2013. — 432 с.

52. Экспозито, Д. Разработка веб-приложений с использованием ASP.NET и AJAX / Д. Экспозито. — СПб.: Питер, 2012. — 400 с.

53. Якоб Нильсен, РалукаБудиу. Как создавать идеально удобные приложения для мобильных устройств. 2013г.