Разработка под iOS. Начинаем

Часть 2

Обзор архитектуры

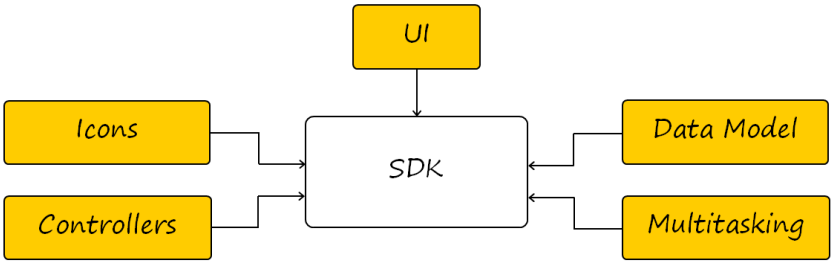
Оглавление

Глава 1

Разработка под iOS. Начинаем

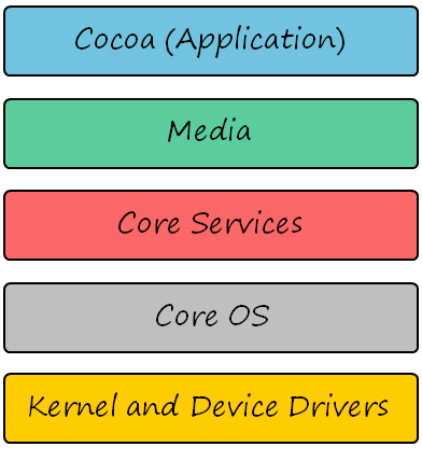
1.1. Базовое представление об архитектуре

На прошлой неделе вы познакомились с языком программирования Swift и попробовали писать на нем свои первые программы.
На этой неделе я познакомлю вас с SDK и инструментами, которыми вам предстоит пользоваться при разработке и отладке своих приложений. Также я познакомлю вас со средой разработки Xcode, особенностями ее настройки. Вместе мы создадим ваш первый проект, соберем и запустим ваше первое работающее приложение.
iOS – это мобильная операционная система для устройств, которые разрабатывает и выпускает компания Apple. Впервые была представлена в 2007 году и стала настоящим прорывом. По сути, iOS стала первой операционной системой для мобильных устройств, ядро которой было почти идентично родстственной операционной системе для персональных компьютеров.
SDK, которое предоставляет Apple, очень многогранно и богато. Кроме того, постоянно пополняется: на конференции WWDC ежегодно представляются новые SDK. Какие из них понадобятся вам при написании приложений, предсказать невозможно – во многом это зависит от того, какое приложение в итоге вы хотели бы получить. На начальном этапе ключевыми для вас, скорее всего, будут UIKit и CoreGraphics, которые позволяют создавать элементы интерфейса.

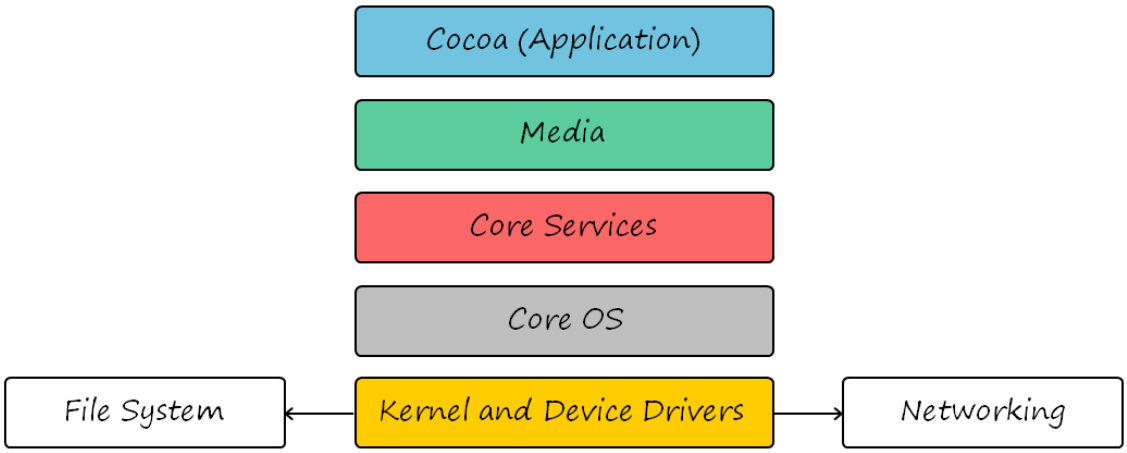


Как большинство многоуровневых операционных систем, iOS представляет собой иерархию слоев:

- уровень аппаратного обеспечения;
- уровень ядра операционной системы;
- уровень системных библиотек;
- уровень прикладных программ.



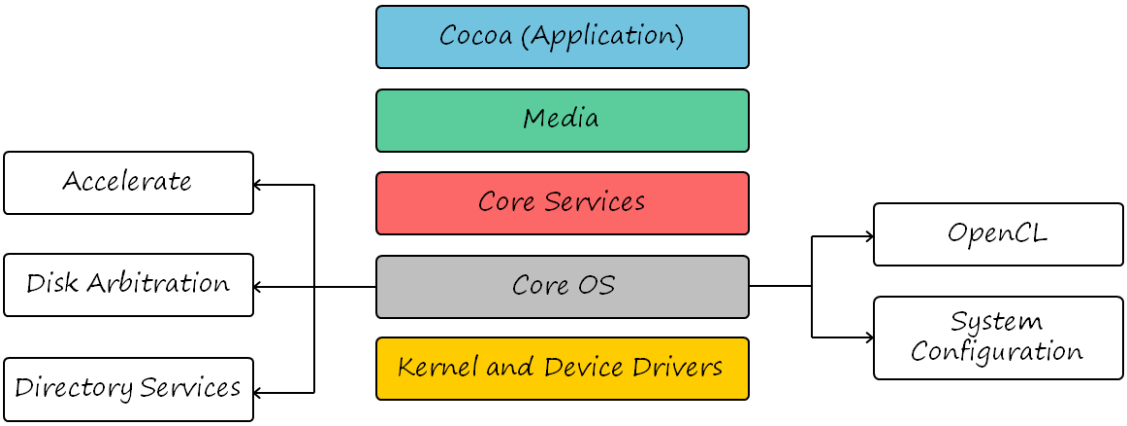
Нижний уровень иерархии – это уровень **Kernel and Device Drivers**. Является основой операционной системы. Отвечает за управление выделением и освобождением памяти, обработку сетевых и других задач операционной системы, а также обработку задач файловой системы. Он напрямую взаимодействует с оборудованием устройства, на котором операционная система запущена.



Следующий уровень – **Core OS** – это уровень ядра операционной системы. Он предоставляет два типа сервисов:

1. низкоуровневые сервисы для работы с оборудованием, сетевого взаимодействия, управлением потоками;
2. высокоуровневые – связанные с безопасностью.

Напрямую со вторым типом во время написания приложений вы взаимодействовать не будете, поэтому более подробно на его компонентах и возможностях мы останавливаться не будем. Отдельно хотелось бы выделить framework Accelerate, который, по сути, представляет собой математическую подсистему – ее можно использовать для выполнения сложных вычислений, в машинном обучении – для запуска обученных моделей.



Следующий уровень – **уровень системных библиотек**. Предоставляет возможность приложениям взаимодействовать с уровнем ядра операционной системы. Библиотеки этого уровня являются базовыми для библиотек всех выше стоящих уровней.
Вот несколько примеров таких библиотек:

1. **Social Media Integration**

   При помощи библиотек Accounts и Social вы можете настроить в своем приложении аутентификацию с использованием аккаунтов, которые уже используются в других приложениях, например, в Twitter или Facebook.

2. **iCloud Storage**

   Позволяет вам хранить произвольные данные пользователя таким образом, чтобы они были доступны на всех его Apple-устройствах.

3. **GCD**

   Предоставляет простой и удобный интерфейс для написания многопоточного кода, распараллеливания выполнения вашим приложением различных задач. Подробнее будем рассматривать в следующем курсе.

4. **Maps**

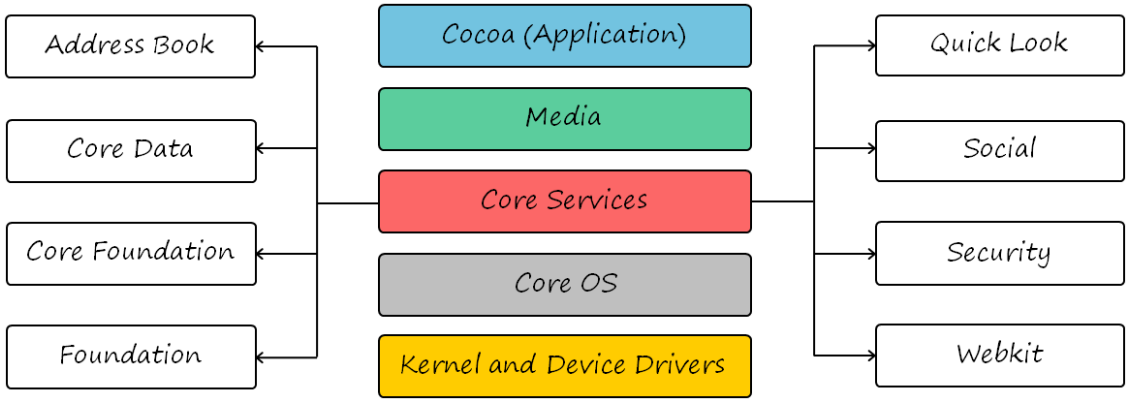
   Позволяет добавить отображение карты, различных объектов и самого пользователя на ней в вашем приложении.

5. **Core Services Frameworks**

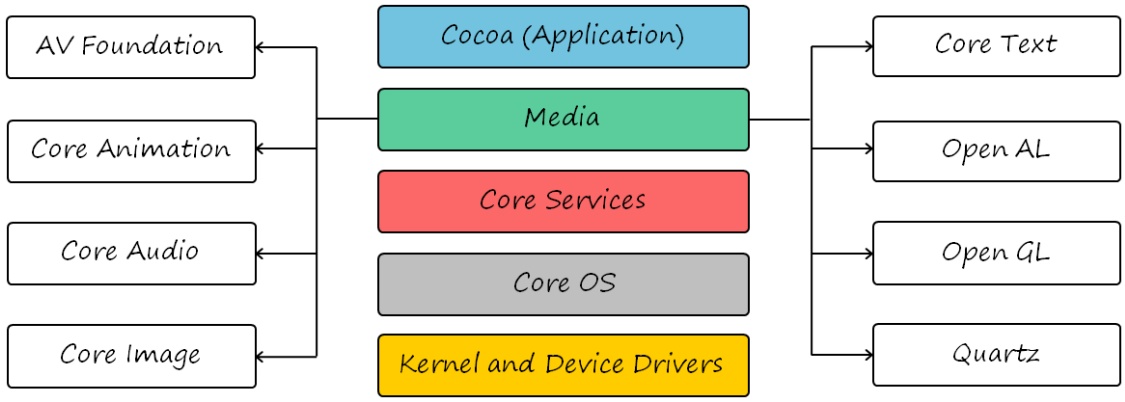
   Около 20 различных библиотек, в том числе:

   - Core Foundation (базовые структуры данных);
   - Core Location (при помощи которой вы можете определять местоположение пользователя);
   - Core Data (которую вы можете использовать для организации хранения больших объемов данных и о которой мы расскажем вам подробнее в следующем курсе).

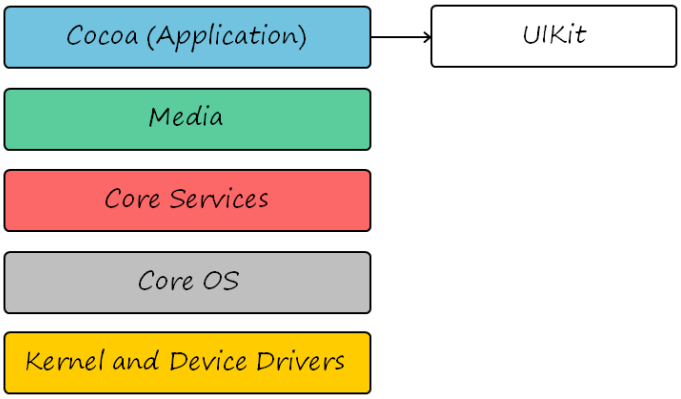
   И многое другое.



Затем идет уровень **Media**. Этот уровень, во-первых, предоставляет вам инфраструктуру для отображения, во-вторых, для работы с динамиком, микрофоном. Служит основой для следующего уровня, на котором, собственно, мы пишем наше приложение. Используя возможности этого уровня, вы можете добавить в свое приложение анимацию, звук, визуальные эффекты, 2D и 3D графику.



Последний уровень – **Cocoa (Application)** – уровень приложений или уровень конкретных прикладных frameworks, которыми вы будете пользоваться чаще всего при создании своих собственных приложений. Уровень, средствами которого создается пользовательский интерфейс вашего приложения. Библиотеки этого уровня могут быть использованы вами для добавления в приложения стандартных элементов управления. Средствами этого уровня обеспечивается возможность для пользователя взаимодействовать с вашим приложением, а для вашего приложения – получать уведомления о тех или иных событиях через Notification Center и многое другое.



1.2. Среда разработки

Итак, мы поговорили о том, какие системные возможности можем использовать для создания приложений. Но кажется, что понятнее, как, собственно, их писать, от этого не стало. Поэтому теперь давайте попробуем создать что-то свое.

Первое, что для этого нам необходимо, – это среда разработки. Конечно, иногда можно писать код и в обычном блокноте, но в большинстве случае это не очень удобно и занимает куда больше времени. В случае с приложениями для iOS это также автоматически означает проблемы с запуском того, что вы написали, так как для тестирования приложений вам понадобится либо iOS-устройство, либо его эмулятор.

На данный момент существует две среды разработки, которые используются iOS-разработчиками, – это Xcode (среда разработки, предлагаемая Apple) и AppCode (его аналог от JetBrains). Но если Xcode предоставляет вам полный набор инструментов для того, чтобы разрабатывать, отлаживать, собирать и запускать ваши приложения, то AppCode может использоваться, по большей части, только непосредственно для написания кода и его отладки. Для того, чтобы собирать приложения, вам понадобиться поставленный параллельно Xcode или хотя бы его часть. Поэтому в дальнейшем в нашем курсе мы остановимся именно на использовании Xcode и его возможностях. Пользоваться лучше самой последней версией среды разработки.

На прошлой неделе вы уже работали в Xcode, когда писали код в playground-е. Теперь давайте попробуем создать в нем ваш первый проект.

Прежде, чем пользоваться средой разработки, полезно настроить ее в соответствии со своими собственными предпочтениями. Для того, чтобы настроить Xcode, вам необходимо открыть его, а затем в меню Xcode выбрать пункт Preferences. Откроются настройки Xcode, в которых вы можете:

1. Настроить общее поведение.

2. Добавить свой аккаунт разработчика, сертификат.

   Это потребуется вам уже на этапе тестирования вашего приложения на реальном устройстве (если оно у вас есть). Информация о том, как это можно сделать, представлена в дополнительных материалах к лекции.

   Подробнее о том, как это можно сделать, можно прочитать здесь:

   https://help.apple.com/xcode/mac/current//dev60b6fbbc7.

3. Настроить параметры тестирования, сборки, запуска (например, можно заставить голосового помощника на вашем маке оповещать вас каждый раз, когда Xcode начинает/заканчивает собирать проект).

4. Настроить управление (short keys, поведение при одиночном, двойном нажатии).

5. Настроить текстовый редактор (например, увеличить/уменьшить шрифт, изменить его цвет, изменить подсветку, цвет фона).

6. Настроить интеграцию с системой контроля версий и многое другое.

Для себя я обычно настраиваю так:

1. На первой вкладке устанавливаю дополнительно к настройкам по умолчанию ContinueBuildingAfterErrors. Это позволит мне видеть все ошибки компиляции после одной попытки сборки. Если не установить этот режим, вам придется пересобирать проект каждый раз после исправления очередной ошибки компиляции, чтобы устранить все проблемы.

2. На второй вкладке добавляю свой AppleID, добавляю аккаунт разработчика.

3. На четвертой вкладке меняю настройку Navigation на Uses Focused Editor – это позволит мне открывать выбранный файл в активном окне редактора (их у вас может быть несколько, я покажу чуть позже).

4. На вкладке Text Editing – добавляю отображение номеров строк, добавляю отображение Page Guide с лимитом, например, в 120 колонок. Это позволит мне сразу заметить, если какая-то строка стала слишком длинной.

Задача: попробуйте поменять разные настройки среды разработки, настроить ее так, чтобы вам было комфортнее работать.

Начать работу с Xcode проще всего с создания своего первого проекта. Давайте попробуем. Итак, мы открыли Xcode, видим, что можно нажать на кнопку "Create a new Xcode project" точно так же, как раньше мы нажимали на кнопку "Get started with a playground".

Давайте сделаем. Нам открылось окно, в котором предлагается выбрать один из существующих шаблонов. Они существуют для того, чтобы разработчикам не приходилось много раз писать своими силами один и тот же код, кочующий из одного проекта в другой.
Мы можем выбрать платформу, для которой хотели бы писать – в нашем случае это iOS, и один из существующих шаблонов. Наибольшую свободу действий вам дает шаблон Single View App – он создает необходимый минимум для работы приложения, но не более того.
Давайте выберем этот шаблон. Дальше мы сможем настроить дополнительные параметры нашего проекта, такие, как, например:

- название;
- название команды разработки;
- название организации.

Все это будет использовано для создания уникального идентификатора вашего приложения и формирования заголовков с copyrights, которые Xcode будет добавлять во все файлы вашего проекта. Также можем настроить язык, на котором нам хотелось бы писать – в нашем случае это будет Swift, указать, хочется ли нам использовать Core Data (в нашем простейшем случае – нет), хочется ли нам дополнительно писать UI и Unit тесты для нашего приложения.

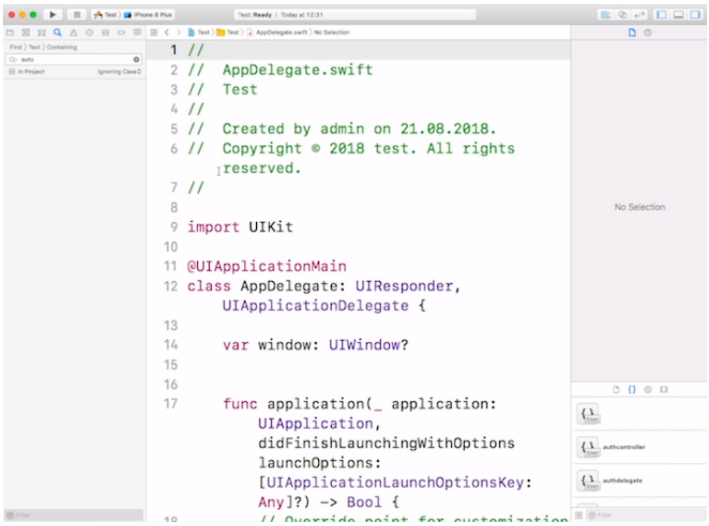
1.3. Что есть в Xcode?

Итак, мы создали проект. Что можем использовать для того, чтобы вносить в него изменения?
Во-первых, слева располагается панель навигации. Здесь вы можете увидеть все файлы вашего проекта или ваших проектов, если их несколько, перейти к детальному просмотру одного из них. Также перейти к тому или иному файлу можно, нажав cmd+shift+O и введя часть названия этого файла. Это более быстрый способ, если точно не помните, где файл расположен (особенно актуально, если проект велик).
Помимо навигационной, в левой части есть еще ряд закладок, которые могут быть вам полезны. Например, вы можете:

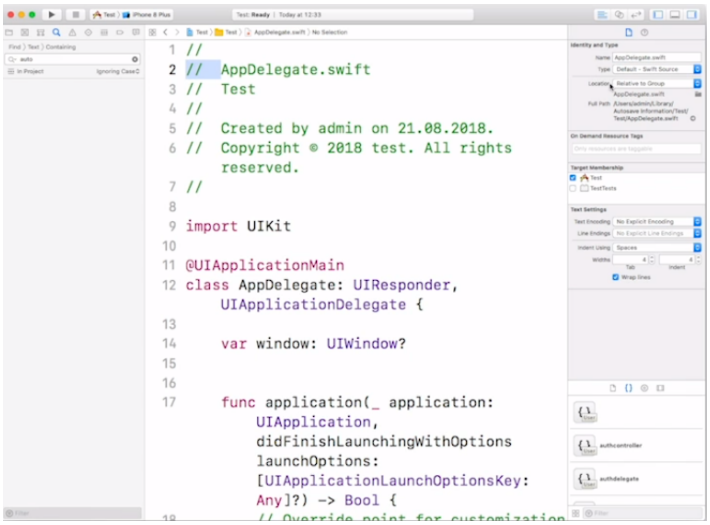
- просматривать список файлов;
- запускать поиск по всему проекту или его частям;
- смотреть список ошибок, которые возникли во время компиляции или запуска приложения;
- просматривать список тестов вашего проекта, запускать их все вместе или по одному, оценивать результат;

- смотреть stack trace во время отладки приложения, смотреть и редактировать список точек остановок, просматривать лог осуществляемых действий (например, сборка, запуск тестов).

В центре располагается окно детального просмотра, здесь можно просматривать и редактировать файлы. Используя кнопки в правой верхней части экрана, вы можете выбрать один из режимов отображения в зависимости от того, что вам необходимо на данный момент.



Первая кнопка – базовый режим, при котором вы видите и редактируете только один файл.
Вторая кнопка – позволяет открыть Assistant Editor и видеть два файла одновременно. При помощи меню в верхней части обоих окон можно выбирать файл, который вы будете видеть в каждом из них. Дополнительно можно в настройках Xcode добавить возможность открывать файлы при помощи cmd + shift + o или выбора файла из списка в том окне, которое на данный момент активно. Использовать такой режим параллельного отображения бывает удобно, когда вы, например, если вы производите рефакторинг старого кода, чтобы видеть одновременно старые и новые файлы.
Assistant Editor можно открывать как справа, так и снизу – в зависимости от того, что вам представляется более удобным.
Третья кнопка будет вам полезна, если вы используете для вашего проекта какую-нибудь из систем контроля версий – она позволяет просматривать историю изменений.
В правой части, как правило, располагается диспетчер свойств выбранного объекта (файла, части интерфейса). У него, как правило, тоже есть несколько закладок, позволяющих просматривать разные группы свойств, редактировать их значения.
Что вы увидите здесь, например, для файла?



Во-первых, file inspector. Он позволяет увидеть имя, тип файла, путь к нему, как абсоолютный, так и относительный (например, относительно группы, в которой он расположен), перейти к файлу (например, используя finder).
Также вы можете посмотреть, к какому target-у относится ваш файл. Обычно несколько таргетов у вас в проекте может быть в том случае, если, например, у вас есть тесты – они существуют в отдельном таргете. Также можно настроить кодировку файла, используемый отступ, для файлов Interface Builder-а здесь же располагаются настройки локализации. Также для файлов Interface Builder-а, используя нижнюю панель, можно просмотреть библиотеку стандартных элементов интерфейса, перетащить часть из них на свои экраны (подробнее об этом будет в лекции про создание интерфейса).
Xcode обычно поставляется не сам по себе, а с некоторой совокупностью инструментов, способных оказать вам существенную помощь при написании, отладке ваших приложений.
Первый и наиболее важный из них, особенно если у вас нет своего устройства на платформе iOS, – это эмулятор. Его можно запустить либо руками, либо просто собрав свой проект для запуска на симуляторе и запустив его. Сделать это можно выбрав в контекстном меню Product→Run, либо используя short keys: cmd + R (собрать и запустить).
Симулятор предоставляет вам возможность отлаживать свое приложение практически как на реальном устройстве, хотя возможности его, конечно, ограничены. Например, вы не сможете использовать акселерометр или тестировать пуш-уведомления на симуляторе. Также именно на симуляторе удобнее всего запускать тесты – примечательно, что он используется для запуска как UI, так и Unit-тестов.
При запуске симулятора вы можете выбрать, какое устройство вы хотите запустить, версию операционной системы, которую вы хотели бы видеть на этом устройстве. Это очень удобно, но следует учитывать, что симулятор все-таки не является реальным устройством и работа вашего

приложения на симуляторе не является гарантией его работы на реальных устройствах. Поэтому перед отправкой приложения на ревью в AppStore и в процессе разработки все же рекомендуется проверять работоспособность приложения на устройстве. Желательно даже не на одном, так как то, что работает на десятом iPhone с iOS 11 совсем необязательно будет работать на пятом iPhone с iOS 8, и наоборот.
Кроме того, в списке полезных для вас инструментов, поставляемых с Xcode, можно назвать Accessibility Inspector, а также Instruments.
Accessibility Inspector позволяет проверять, насколько ваше приложение удобно для людей с ограниченными возможностями. Также он может быть полезен при написании UI-тестов.
Instruments используются для профилирования вашего приложения, улучшения его производительности. Об инструментах мы еще поговорим подробнее чуть позже. Кроме того, вы сможете плотно поработать с ними во время выполнения домашнего задания.

1.4. Пишем код, исправляем ошибки

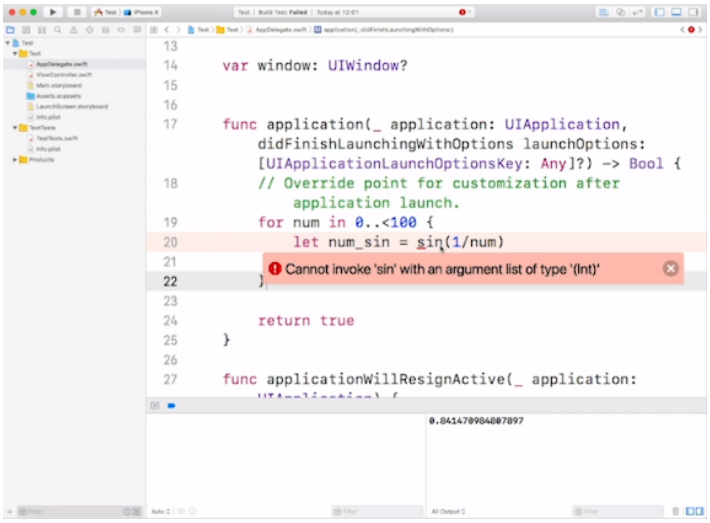
Вернемся к нашему проекту. Какие файлы там уже есть?
Во-первых, нас интересует файл AppDelegate. Этот класс отвечает за взаимодействие вашего приложения с операционной системой. В том числе там находится метод applicationDidFinish LaunchingWithOptions, который вызывается после завершения запуска приложения.
Обычно этот метод используется для выполнения действий, необходимых для дальнейшей работы в приложения. Но на начальном этапе, когда, по сути, у вас нет еще приложения как такового, этот метод отлично подходит для того, чтобы экспериментировать.
Давайте что-нибудь здесь напишем и запустим наше приложение. Например, вот так:

```
let num = 1.0 // Заведем переменную num со значением 1.0
print(sin(1 / num)) // Выведем синус числа, обратного к num
```

Запустим. Для этого мы можем воспользоваться опцией Run в контекстом меню Product или же комбинацией клавиш Cmd + R.
Предварительно выберем устройство, на котором хотели бы запустить приложение. Возьмем, например, симулятор десятого iPhone.
Первый раз запущу с использованием контекстного меню, чтобы просто показать его вам. В дальнейшем буду пользоваться горячей комбинацией клавиш (cmd + R), так что не пугайтесь, когда проект магическим образом будет начинать собираться и запускаться.
Замечательно. Мы вывели синус 1. В нижней части экрана, в консоли, можно его увидеть.
Давайте немного исправим код, чтобы выводить синусы чисел, обратных к числам от 0 до 100.

```
for num in 0..<100 { // напишем цикл от 0 до 100
    let num_sin = sin(1 / num) // переменной num_sin присвоим значение синуса
    единицы, деленной на num print(num_sin) // затем выведем
    это значение
}
```

Попробуем запустить. Не получается – видим ошибку, выглядит она вот так. Если нажмем на нее – можем посмотреть подробную информацию.



Иногда полное описание ошибки не вмещается в одну строку, но в данном случае это не так, поэтому ничего не происходит.
Давайте исправим.

```
for num in 0..<100 {
    let num_sin = sin(Double(1 / num)) // Приведем аргумент к типу Double
    print(num_sin)
}
```

Отлично, ошибка пропала. Теперь мы снова можем запустить наше приложение. Ожидаем, что в консоли будут выведены синусы чисел, обратных к числам от одного до ста. Но этого не происходит – выполнение почему-то останавливается.
Нам удалось скомпилировать и запустить наш проект. Но теперь ошибка произошла уже во время работы приложения. Ее мы тоже можем увидеть, причем посмотреть на нее можем сразу с нескольких сторон.
Во-первых, в нижней части экрана вы можете увидеть отладочное окно. Здесь вы можете по-

смотреть список переменных и их текущие значения. Также у вас есть консоль, на которую выводится отладочная информация. С консолью вы также можете взаимодействовать при помощи набора команд. Об отладке подробнее расскажу чуть позже.
Пока же давайте просто разберемся с нашей ошибкой. Справа вы можете увидеть текущее состояние загрузки CPU, используемую память, скорость работы с диском, скорость обмена информацией по сети. Также здесь вы можете увидеть stack trace всех потоков, которые на данный момент созданы. В частности, того потока, в котором случилась ошибка. Именно он будет для вас развернут.
Мы можем перемещаться по стеку вызовов, просматривать код методов, в том числе и ассемлерный вариант. Ошибка и ее описание будут рядом с той инструкцией, которая привела к этой ошибке.
Итак, мы видим, что ошибка случилась из-за того, что написанные нами команды привели к осуществлению деления на ноль. Действительно, первым значением переменной num будет 0, поэтому при выполнении вычисления синуса от 1, деленной на num, мы получаем ошибку.
Давайте исправим, начнем итерацию с 1.

```
for num in 1..<100 {
    let num_sin = sin(Double(1 / num))
    print(num_sin)
}
```

Замечательно, теперь все корректно работает, в консоли мы можем увидеть вычисленные значения.
На самом деле не очень интересно – почти всегда значение нулевое. Чтобы завершить работу приложения, можно использовать то же контекстное меню, что и для запуска, чуть ниже есть команда stop. Или сочетание клавиш cmd + .
Давайте теперь попробуем установить breakpoint внутри цикла.
Запустим снова – теперь мы остановились ровно там, где и хотели. Снова видим список переменных, видим их значения. Снова есть консоль, в которой это значения будут выводиться. Для отладки можно использовать специальные кнопки отладки. Они позволяют вам

1. сделать неактивными точки останова (все сразу);
2. продолжить выполнение программы;
3. перейти к следующей инструкции;
4. провалиться внутрь функции;
5. вернуться из функции.

В общем, полный набор необходимых команд. В консоли вы также можете увидеть вывод. Кроме того, вы можете использовать консольные команды po (print object – для того, чтобы просмотреть значение какой-то переменной) и exec (чтобы попробовать выполнить какой-то код).

1.5. Структура проекта, настройка схемы и таргета

Давайте теперь посмотрим на структуру нашего проекта. Например, в finder-е. У нас есть, как правило, Xcodeproj. Выглядит, как файл, но на самом деле это не файл. Это пакет mac os, в котором скрыто много всего. Посмотреть на содержимое можно нажав на нем правой клавишей мыши и выбрав "Показать содержимое пакета". Внутри есть файл pbxproj – по сути, описание вашего проекта, то, как он выглядит в навигаторе проекта. Если вам нужно к одному проекту подцепить какие-то другие (например, стороннюю библиотеку), вам нужен xcworkspace. Еще есть папка xcuserdata – внутри вы можете найти свои локальные настройки. Например, файл с описаниями точек останова, описание схем.
Помимо Xcodeproj внутри папки вашего проекта, как ни странно, будут еще файлы вашего проекта.
Например, если мы провалимся в нашем случае в папку с именем проекта, мы сможем увидеть и наш первый view controller, и app delegate. Также тут есть:

- plist-файл с дополнительными настройками вашего проекта, обычно он называется Info.plist;
- проект локализации – lproj, внутри можно увидеть локализуемые файлы, пока внутри только storyboard-ы – особые файлы, описывающие взаимоотношения между различными элементами вашего интерфейса;
- также asset-ы, в которых вы можете хранить ресурсы вашего приложения, например, картинки или цвета.

Вы можете увидеть, собственно, ваш проект – при нажатии на него можете редактировать параметры проекта или конкретного таргета. Здесь есть, например, настройки компиляции, на вкладке Info – настройки локализации, доступные конфигурации, минимальная версия операционной системы, на которой ваше приложение будет запускаться.
Внутри одного проекта у вас может быть несколько таргетов и несколько схем. Давайте разберемся, что это такое.
Таргет – это отдельный продукт, который и будет собираться. Посмотреть на него можно в папке Products. К конкретному таргету линкуются файлы, которые в рамках этого таргета будут собираться. Несколько таргетов у вас в проекте может быть, например, в том случае, если в вашем проекте есть тесты или какие-либо extension-ы, например, виджет.

Схема – это настройки того, как тот или иной таргет будет собираться, тестироваться, запускаться. Схем может быть несколько, схемы могут быть как локальными (существующими только для вас), так и общими (актуально, если вы не единственный разработчик, работающий над проектом). Параметры запуска, параметры отладки – все это задается через схему.
Если открыть редактирование схемы, можно увидеть:

1. Параметры сборки (например, хотим ли мы, чтобы сборка была распараллелена, или для чего можем собирать с этой схемой – для анализа, запуска, тестирования, профилирования);
2. Параметры запуска:
   - в какой конфигурации будем запускать - здесь сейчас есть Debug и Release;
   - какой файл является исполняемым;
   - параметры диагностики – удобно использовать на этапе разработки приложения, например, можно включать MainThreadChecker, который позволит определить, если что-то, что должно делаться только на главном потоке, делается в одном из побочных;
3. Можно настроить параметры тестирования (указать, например, какие тесты хотим запускать, какие нет);
4. Профилирования (какой бинарник запускаем, в какой конфигурации);
5. Анализа;
6. Сборки для стора;
7. Различные параметры эмулятора, например, параметры геолокации и языка.

Давайте посмотрим, какие вкладки есть для настройки таргета.

General:

1. Редактировать идентификатор вашего проекта, версию, номер сборки.
2. Для того, чтобы для любого приложения для iOS и Mac OS можно было однозначно определить разработчика, при сборке ваше приложение будет подписываться с использованием сертификата разработчика. Это позволяет обеспечивать большую безопасность пользователям при установке разного рода приложений, например, показывать ему сайт разработчика и предлагать подтвердить желание установить приложение, в том случае если разработчик не является доверенным. На второй вкладке вы можете выбрать сертификат, которым собираетесь подписывать ваше приложение.

3. Можем указать, на каких устройствах хотели бы, чтобы запускалось наше приложение. Минимальная версия операционной системы редактируется на уровне всего проекта, поэтому на уровне таргета редактирование недоступно. Но мы можем, например, указать, что наше приложение можно запускать только на iPhone – на iPad оно в этом случае будет запускаться в режиме совместимости. Также можем указать storyboard, который будет стартовым для нашего приложения, поддерживаем мы или нет горизонтальный/вертикальный режим (довольно частой является ситуация, при которой поддерживается только вертикальный режим – в этом случае при повороте устройства в горизонтальный режим поворота интерфейса не происходит).

4. Можно выбрать, какой файл будем использовать в качестве иконки приложения, какой файл будем использовать в качестве launch screen – иными словами, что будет видеть пользователь, пока приложение запускается.

5. Можем просмотреть список присоединенных библиотек, можем его редактировать.

Capabilities – можем настроить, что умеет наше приложение. Например, здесь мы можем сказать, что в нашем приложении будут пуш-уведомления.
Info – удобное редактирование файла Info.plist. Здесь можно также, например, указать файл, который будет стартовым; указать файл, который будет использоваться в качестве экрана загрузки и многое другое.
Можно отредактировать настройки сборки. Например, здесь можно указать версию Swift, которую вы хотите использовать. Можно отредактировать список стадий сборки, например, добавить выполнение какого-то своего скрипта.
Дальше, внутри папки нашего проекта вы можете увидеть файлы, на которые мы уже смотрели ранее в finder. Вот, например, файл Info.plist, можем перейти к нему, отредактировать. На файл AppDelegate мы уже смотрели, с тем, что такое storyboard и view controller, будем подробнее разбираться в следующей лекции.

1.6. Профилирование

Что есть для анализа того, что написали?
Итак, небольшой кусочек кода в нашем проекте мы уже написали (помните цикл в applicationDid FinishLaunchingWithOptions?), попробовали использовать встроенный отладчик для того, чтобы искать ошибки.
Что еще у нас для этого есть?
Во-первых, в Xcode есть встроенный статический анализатор. Использовать его можно, выбрав в контекстном меню Product пункт Analyze. Может помочь найти проблемные места, напри-

мер, какие-то участки кода, которые могут привести к непредсказуемому поведению или крэшу. Бывает полезно, пользуйтесь.
Далее – у вас есть ошибки и предупреждения, которые генерируются во время компиляции. Такую ошибку мы уже видели. Если в вашем приложении есть ошибки компиляции – до запуска дело даже не дойдет, вам придется с этими ошибками разобраться. Если у вас есть предупреждения, собрать и запустить у вас получится, но, тем не менее, лучше стараться, чтобы и предупреждений на этапе компиляции у вас не было – их отсутствие позволяет предупредить проблемы, также ускоряет компиляцию. Достаточно распространенный тип предупреждений – это предупреждения, которые появляются у вас в том случае, если вы решили поднять deployment target и где-то до этого пользовались старыми (deprecated) методами для поддержки старых версий операционной системы.
Следующий вид ошибок – это runtime issues. Это ошибки, которые нашел отладчик во время работы вашего приложения. Здесь может свои уведомления писать, например, main thread checker – помните, я показывала вам, что можно включить его в настройках схемы, чтобы контролировать выполнение определенных инструкций на главном потоке? – или другие runtime профилировщики.

Задача: включить main thread checker и придумать, какой код нужно добавить в applicationDid FinishLaucnhing, чтобы он остановил выполнение.

Дополнительный и очень полезный инструмент – это Debug View Hierarchy. Если вы запустили ваше приложение и видите не совсем то, что ожидали, при этом не понимаете, в чем же дело и почему так случилось, вы можете вызвать view debugger и посмотреть на ваш интерфейс "в разрезе", проинспектировать отдельные элементы, их свойства, посмотреть, какие constraints (это условия, которые определяют расположение элементов на экране, как их использовать, вы узнаете уже в следующей лекции) активны, какие нет, – словом, более подробно изучить интерфейс, чтобы понять, где и что пошло не так.
Похожий инструмент есть и для того, чтобы контролировать выделение и утечки памяти, он называется Memory Graph Debugging.

Задача:придумать пример ошибки, которая будет выявлена memory profiler-ом.

Также можно использовать внешние по отношению к Xcode инструменты – те самые, которые поставляются с ним в комплекте. Чтобы запустить, надо в контекстном меню Product выбрать Profile. Откроется окно выбора инструмента. Нас будут интересовать два – Leaks и Time Profiler. Leaks – используется для того, чтобы искать утечки памяти. Запускаем, некоторое время работаем с нашим приложением, получаем запись его работы, которую можем проанализировать. Можем проанализировать для каждого объекта изменение счетчика ссылок. В том случае, если

в приложении есть утечки – мы их увидим.

Задача: вы уже готовили пример кода, который не нравился Memory profiler-у. Добавьте его в свое приложение и убедитесь, что при помощи инструмента Leaks находятся утечки. Исправьте проблему, повторите процесс профилирования – убедитесь, что утечек больше нет.

Time Profiler позволяет найти просадки производительности вашего приложения – например, определить участки кода, которые выполняются дольше других. Бывает очень полезно. Работает так же, как Leaks – вы запускаете приложение при помощи Profile, затем выбираете Time Profiler. Выглядит он следующим образом.
Аналогично запускаете запись, работаете какое-то время с приложением, потом – или во время – можете проанализировать использование CPU, посмотреть, какие инструкции выполняются дольше других.

Задача: проанализируйте с использованием Time Profiler, вычислите строку, которая выполняется дольше других, попытайтесь объяснить, почему.

```
for num in 1..<100000 {
    let num_sin = sin(Double(1 / num))
    NSLog("%f", num_sin)
}
```

Также в Xcode есть инструменты для написания, запуска тестов – UI и Unit-тестов. Писать тесты здорово – они помогают вам предусмотреть больше возможных вариантов входных данных, на которых ваш код может начать работать не так, как вы ожидаете. Также позволяют отловить больше ошибок в будущем, при изменении кода, на ранних этапах.
Стандартным средством, предлагаемым Apple для написания тестов, являются framework-и XCTest, XCUITest. В них есть полный набор инструментов для написания тестов, в частности, стандартный набор assert-ов для осуществления разного рода проверок.
После того, как тесты завершились, можно просмотреть отчет о выполнении тестов – для этого достаточно нажать на конкретном тесте правой кнопкой, перейти к просмотру отчета. Особенно актуально, если тест не прошел: можно по шагам посмотреть, как выполнялся тест, в какой момент он упал и почему так случилось.

1.7. Сторонние библиотеки

Наконец, несколько слов о том, как вы можете использовать в своем приложении сторонние библиотеки. Разумеется, вы всегда можете это сделать и переиспользование кода – это хорошо. Но

хотелось бы настоятельно порекоммендовать вам несколько раз подумать, прежде чем использовать в своем приложении ту или иную библиотеку, ведь все возможные ошибки в ней будут восприниматься пользователем как ваши ошибки. Поэтому, да, переиспользовать код хорошо, но только в том случае, если вы в этом коде уверены.
Отойдя от темы, использовать или нет сторонние библиотеки, обратимся к тому, как это можно сделать.

1. Вариант номер один – затянуть библиотеку исходниками себе в проект. Этот вариант в принципе прост и понятен, при этом достаточно часто используется, так как дает полный контроль над кодом, который вы используете в своем проекте. Но имеет свои недостатки. В частности, в дальнейшем библиотеки сложно будет обновлять, сложно будет гарантировать, что библиотеки не были ошибочно модифицированы, после тех или иных модификаций по-прежнему работают корректно. Поэтому останавливаться подробнее на нем не будем.

2. Второй вариант – использовать один из пакетных менеджеров. Использование пакетного менеджера существенно упрощает добавление зависимости в проект, актуализацию этой зависимости.

Для целей нашего курса будем использовать CocoaPods. Он удобен и прост в использовании, кроме того, большая часть библиотек CocoaPods поддерживают. Подробнее об этом пакетном менеджере можно почитать, перейдя по ссылке в дополнительных материалах к лекции.
Говоря о пакетных менеджерах, CocoaPods – это не единственный вариант. Полный список можно также посмотреть в приложении к лекции.
Для того, чтобы использовать cocoapods, нам прежде всего нужно установить его на наш компьютер. Cocoapods – пакетный менеджер, написанный на языке Руби. К счастью, на компьютерах с операционной системой Mac OS все необходимое для запуска программ на Руби есть. Версия Руби, которая установлена на компьютерах с Mac OS, является совместимой с CocoaPods. Поэтому в том случае, если вы не меняли версию Руби на своем компьютере, проблем при установке возникнуть не должно.
В том же случае, если вы версию Руби все же меняли, и установленная на вашем компьютере версия оказалась несовместима с CocoaPods, вы можете использовать Ruby version manager, чтобы изменить версию на подходящую.
Ссылка на подробную пошаговую инструкцию по установке CocoaPods представлена в дополнительных материалах к лекции. Также в дополнительных материалах есть ссылка на инструкцию по установке конкретной версии Руби на ваш компьютер – в том случае, если возникнут проблемы.
Затем мы хотим добавить все необходимое для использования cocoapods в наш проект. Для этого воспользуемся в директории нашего проекта командой pod init. После этого закроем наш проект – в дальнейшем нам нужно будет его открывать, используя уже workspace.

В cocoapods нас будут интересовать 2 файла:

- во-первых, podfile. Это файл, в котором вы можете указать все зависимости, которые хотите добавить. Указывается в таком виде:

```
pod 'CorePlot', '~> 2.2'
```

- во-вторых, podfile.lock. Это файл, в котором описаны установленные в проекте библиотеки, их версии и подзависимости этих библиотек. Генерируется после выполнения команды

```
pod install
```

  в первый раз.

При дальнейших выполнениях этой команды для установки зависимостей будут подтягиваться именно версии зависимостей из podfile.lock, если только явно не указывается другая версия в podfile. Также версии обновляются, если выполнить команду

```
pod update
```

После выполнения pod install в первый раз в вашем workspace появляется проект Pods, внутри него вы можете увидеть podfile и все подтянутые зависимости. Теперь их можно использовать. И теперь нужно открывать Project_name.workspace, а не Project_name.xcodeproj.
Workspace – это документ Xcode, который позволяет объединить несколько проектов таким образом, чтобы с ними можно было работать совместно. Открывая workspace вместо xcodeproj, вы сможете видеть не только ваш проект, но и, в нашем случае, проект Pods.
Важным моментом стоит упомянуть директиву use_frameworks!.
Xcode поддерживает два типа линковки зависимостей:

- Статический, при котором зависимости становятся частью вашего приложения, частью вашего бинарника;
- Динамический, при котором зависимости собираются в виде отдельных динамических библиотек (dylib), они затем будут подключаться к вашему проекту на этапе запуска.

До Xcode 9 библиотеки на Swift можно было использовать только как динамические библиотеки. Сейчас можно использовать статическую линковку и для библиотек на Swift. Для этого вам нужен будет Xcode версии больше или равной девятой и CocoaPods версии больше или равной 1.5.0. Так как использование динамических библиотек увеличивает время запуска приложения, использовать их не рекомендуется, особенно если зависимостей в вашем проекте много.

Давайте попробуем добавить какую-нибудь зависимость, например, библиотеку для отрисовки графиков – CorePlot. Для этого для начала сделаем pod init в папке нашего проекта. Затем отредактируем podfile.

```
target 'Test' do
pod 'CorePlot', '~> 2.2'
end
```

В нем укажем, что у нас есть таргет Test, у которого будет зависимость CorePlot. Версия выбрана таким образом, чтобы библиотека была на Objective C и нам не пришлось использовать use_frameworks! даже в том случае, если у нас не Xcode 9 и CocoaPods версии, меньше 1.5.0.
После этого необходимо выполнить команду pod install. При ее выполнении происходит анализ podfile и podfile.lock (если уже есть), устанавливаются все необходимые зависимости. После этого закрываем проект и открываем workspace, чтобы увидеть наши зависимости.
Для того, чтобы обновить версию библиотек в дальнейшем, можно использовать команду pod update.
Итак, мы создали в Xcode проект своего первого приложения. Попробовали писать в нем свой первый код, отлаживать его, искать и исправлять в нем ошибки, анализировать его производительность. Также я показала вам, как можно добавить в ваш проект зависимость, используя пакетный менеджер CocoaPods. На следующей неделе вы продолжите работать с созданным проектом и научитесь создавать элементы интерфейса.