Intro

Tools

- В качестве IDE используется Xcode или AppCode
- Toolchain идет вместе с Xcode
- Компилятор Clang (Front-End для компилятора LLVM) для преобразования кода в LLVM IR (intermediate representation)
- В Xcode есть симулятор, а это значит, что на компьютере мы запускаем не такой-же код, как на устройстве (другая архитектура, другие библиотеки, …)

Languages

- C
- C++
- Objective-C
- Objective-C++
- Swift (использьзует не Clang, а свой Front-End)

Runtime

- Специальная библиотека, которая используется компилятором для того, чтобы реализовать функции встроенные в язык программирования

- Включает в себя: стандартные типы языка, стандартные функции языка, механизмы управления памятью, …

Bundle

- Папка с стандартизированной иерархической структурой, которая содержит выполняемый код и ресурсы используемые этим кодом

Frameworks

- Framework — это bundle с заготовочными файлами
- Могут быть динамическими/статическими библиотеками (lib, dylib)
- Официально, до iOS 8 dylib созданные пользователем не поддерживаются
- Все системные — динамические
- Содержат весь API (UI, сеть, карты, работа с файлами,…)
- Большинство фреймворков имеют интерфейсы только для ObjC и Swift

IPA

- ZIP архив с bundle
- В bundle есть XML с метаданными (файл Info.plist)
- Выполняемый код — подписан
- Для каждой архитектуры свой код (armv7, armv7s, arm64)
- Для приложений на Swift содержит динамические библиотеки с Swift Runtime, что увеличивает размер IPA. ObjC Runtime интегрирован в iOS.

Executables

- Используют Mach-O для хранения исполняемого кода одной архитектуры
- Если архитектур несколько, то они объединяются при помощи FAT (просто склейка нескольких архитектур)

Info.plist

Метаданные приложения включающие в себя:

- Уникальный идентификатор приложения
- Версию приложения
- Device capabilities (WIFI, Camera, GPS, …)
- Путь к Launch Images
- Privacy (доступ к камере, микрофону, GPS, …)

Codesign

- На устройстве можно запускать только подписанный код
- Каждая архитектура Mach-O подписывается отдельно

Codesign

- Приватный ключ
- Сертификат (публичный ключ + мета информация) подписанный AWDRCA (Apple Worldwide Developer Relations Certification Authority)
- Провизионинг (файл с расширением mobileprovision)

Provisioning

- XML подписанный AWDRCA
- Определяет, на каких устройствах код может быть выполнен (хранит список UUID устройств)
- Определяет, кем код может быть подписан (хранит список сертификатов разработчиков)
- Определяет, что код может делать (работать в фоне, отслеживать локацию пользователя, поддерживать подключение отладчика, …)

Provisioning

- Development (test push notifications, можно подключать отладчик, ограниченный набор девайсов)

- AdHoc (prod push notifications, ограниченный набор девайсов)

- In House (prod push notifications, не ограниченный набор девайсов)

- App Store (prod push notifications)

ObjC

ObjC

- Существуют Objective-C и Objective-C++
- Для компиляции используется Clang
- Все ключевые слова начинаются с @
- Грубо можно сказать, что код на ObjC транслируется в код на C++

```
clang -rewrite-objc file_name_of_the_source_code
```

Runtime

- Живет в shared library libobjc.A.dylib
- Добавляет объектно ориентированные свойства для C
- С его помощью код на C/C++ может взаимодействовать с кодом на ObjC

Foundation

- Этот фреймворк определяет базовый слой для всех Objective-C классов

- Содержит классы коллекций, строк, ….

Dynamic Binding

- На этапе компиляции не известно, какой метод будет вызван. Весь роутинг осуществляется на этапе выполнения

- Главное преимущество и в тоже время недостаток ObjC

Prefixes

- У Obj-C нет namespaces
- Для классов принято использовать префиксы
- Примеры: CLLocationManager, NSObject, NSString, …
- В настоящее время Apple рекомендует использовать трехбуквенные префиксы

Entities

- Class
- Class Instance
- Protocol
- Categories

Class

```
@interface <#class name#> : <#superclass#>

@end
```

```
@interface SimpleClass : NSObject

- (void)doSomethingWithObj1:(NSObject *)obj1 obj2:(NSObject *)obj2;

@end

@implementation SimpleClass

- (void)doSomethingWithObj1:(NSObject *)obj1 obj2:(NSObject *)obj2
{
    NSLog(@"obj1: %@ obj2: %@", obj1, obj2);
}

@end
```

Root Class

- Базовый класс, который определяет интерфейс и поведение общее для всех объектов наследованных от него
- Есть два публичных NSObject и NSProxy
- Почти всегда (99.99%) класс является потомком NSObject

Methods

```
@interface SimpleClass : NSObject

- (void)doSomethingWithObj1:(NSObject *)obj1 obj2:(NSObject *)obj2;

+ (void)doSomethingWithObj1:(NSObject *)obj1 obj2:(NSObject *)obj2;

@end
```

- Есть Class Methods (+) и Instance Methods (-)
- Не имеют модификаторов видимости (public, private, …)

Categories

```
@interface NSObject (Extension)

+ (void)printHello;

@end

@implementation NSObject (Extension)

+ (void)printHello {
    NSLog(@"Print hello");
}

@end
```

- Позволяет добавить методы к существующему классу
- Может быть анонимной ()

Ivars

```
@implementation SimpleClass {
@public
    int n;
@private
    NSString *str;
@protected // default
    bool b;
@package
    float f;
}
```

Properties

```
@interface SimpleClass : NSObject

@property NSString *str;

@end
```

```
SimpleClass *sc1 = [SimpleClass new];
sc1.str = @"some text";
NSLog(@"%@", sc1.str);
[sc1 setStr:@"some text2"];
NSLog(@"%@", [sc1 str]);
```

- @property — “синтаксический сахар”
- Создает геттер, сеттер и ivar
- Геттер и сеттер можно переписать
- Если переписать геттер и сеттер, то ivar создан не будет
- Можно задать имя ivar при помощи @syntesize

Protocol

```
@protocol SimpleProtocol <NSObject>

+ (void)printDescription;

@end

@interface SimpleClass : NSObject <SimpleProtocol>

@end
```

- Определяет интерфейс, который должен реализовать класс

Messages Sending

[receiver message];
[[SimpleClass new] doSomethingWithObj1:@“1” obj2:”2”]

- nil — указатель на “нулевой” объект
- nil можно слать любые сообщения
- Сообщения можно слать классам и экземплярам класса

Self, Super

```
- (NSString *)description {
    return [super description];
}

- (void)printDescription {
    NSLog(@"%@", [self description]);
}
```

Class instance

```
SimpleClass *sc1 = [SimpleClass new];
SimpleClass *sc2 = [[SimpleClass alloc] init];
```

- alloc — выделяет память под объект
- init — вызов конструктора по умолчанию
- new эквивалентно вызову [[SimpleClass alloc] init]
- new/alloc/init определены у NSObject
- при удалении объекта вызывается dealloc

Memory Management

- MRC (Manual Reference Counting) или MRR (Manual Retain Release)

- ARC (Automatic Reference Counting)

MRC

- Вы владеете объектом если вызвали методы, которые начинаются с alloc, new, copy, mutableCopy (например allocMyObject, но не allocate)

- Если вы хотите заводить каким-то объектом (продлить его жизнь), то должны вызвать retain

- Когда объект больше не нужен, вы должны его освободить при помощи release

AutoreleasePool

- Если у объекта вызвать autorelese, то он будет помещен в AutoreleasePool

- Объекты помещенные в AutoreleasePool живут до тех пор, пока у AutoreleasePool не будет вызван метод drain.

ARC

- Сам расставляет retain/release/autorelease

- __strong, __weak, __unsafe_unretained, __autoreleasing

Exceptions

```
@try {
        [SimpleClass printHello];
    }
    @catch(NSException * ex) {
        NSLog(@"Exception caught: %@", ex);
    }
    @catch(id ex) {
        NSLog(@"Unknown exception caught");
    }
    @finally {

    }


+ (void)printHello {
    NSException * exc = [NSException
exceptionWithName:NSInternalInconsistencyException
                                                reason:@"Not implemented."
                                              userInfo:nil];
    @throw exc;
}
```

Blocks

```
[scoreReporter reportScoreWithCompletionHandler:^(NSError *error) {
} }]
```

- Анонимная функция

- http://fuckingblocksyntax.com/

NSObject Protocol

- Определяет набор фундаментальных методов для всех Objective-C объектов в том числе методы для подсчета ссылок (retain/release/autorelease/retainCount)

NSObject Class

- Содержит методы для создания объектов alloc/new/copy/mutableCopy

NSString

- NSStinrg *tmp = @“some text”;

NSNumber

- NSNumber *num = @(0.3f);

- NSNumber *b = @(YES);

Collections

- NSArray / NSMutableArray
- NSDictionary / NSMutableDictionary
- NSSet / NSMutableSet
- NSOrderedSet / NSMutableOrderedSet
- CF коллекции (CFDictionary, …)

NSArray

- NSArray *tmp = @[someObj1, someObj2];

NSDictionary

- NSDictionary *tmp = @{ key1Obj: value1Obj, key2Obje: value2Obj]

KVC

- Неформальный протокол NSKeyValueCoding
- Определяет механизм, который позволяет работать со свойствами по имени
- - (id)valueForKey:(NSString *)key
- - (void)setValue:(id)value forKey:(NSString *)key

KVO

- Неформальный протокол NSKeyValueObserving
- Определяет механизм, который позволяет объекту получать уведомления о изменениях свойств других объектов
- - (void)addObserver:(NSObject *)anObserver forKeyPath:(NSString *)keyPath options:(NSKeyValueObservingOptions)options context:(void *)context
- - (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change context:(void *)context

Links

- Objective-C in Wikipedia
- Cocoa Core Competencies
- Programming with Objective-C
- The Foundation Framework
- Хрестоматия iOS паттернов