# Objective-C

Главный инструмент для разработки под iOS

# Литература и справочные материалы

- Б. Керниган, Д. Ритчи: "Язык программирования Си".
- С. Кочан: "Программирование на Objective-C", 6-ое издание.
- https://developer.apple.com/
- http://nshipster.com/
- http://www.raywenderlich.com/
- http://www.objc.io

# Свойства языка программирования Objective-C

- Объектно-ориентированный ЯП.
- Типизация: и статическая, и динамическая.
- Надмножество языка C.
- Используется для разработки под Mac OS и iOS.
- Компиляторы: Clang и GCC.

# ObjC на низком уровне

- Транслируется в промежуточный C-код, затем компилируется в машинный.
- Зависит от Objective-C runtime (C-шная среда выполнения).
- За счет runtime позволяет динамически создавать классы и методы в момент выполнения.

# ООП возможности

- Классы и экземпляры классов.
- Наследование (не множественное).
- Полиморфизм (за счет динамических особенностей).
- Инкапсуляция переменных экземпляра.
- Нет абстрактных классов, но есть “протоколы”.

# Типы в Objective-C

Наследие C (по умолчанию создаются на стэке):

- Скалярные: char, short, int, long, long long (включая unsigned), double, float, а также void.
- Пользовательские: enum, union.
- Составные: struct.

ObjC (создаются в куче, всегда указатели):

- Классы и протоколы.
- id - универсальный тип (= “любой” класс).

# Классы

Описываются методами и свойствами.

```
#import <Foundation/Foundation.h>
// Объявление класса MyClass
@interface MyClass : NSObject // Наследуется от NSObject
@property int myProperty; // Свойство типа int
// Описание метода с двумя параметрами
 -  (void)myMethodWithParam1:(int)p1 andParam2:(id)p2;
@end // Конец объявления
```

```
// Реализация класса
#import “MyClass.h”
@implementation MyClass {
    int _myIvar; // Переменная экземпляра
}
-(void)myMethodWithParam1:(int)p1 andParam2:(id)p2 {
    // Какая-нибудь логика выполнения (для примера)
    if (p2 == nil) // nil - нулевой указатель
        return;

    if (p1 == self.myProperty) { // Обращение к свойству
        _myIvar += p1; // Обращение к переменной экз.
        printf(“%d”, _myIvar);
    }
}
@end
```

# Экземпляры класса

Создается объект - экземпляр определенного класса. Этому объекту посылается сообщение.

```
#import “MyClass.h”
…
MyClass *myObject = [[MyClass alloc] init]; //Объект в стэке
myObject.myProperty = 1; // Изменяем свойство
// Вызываем метод с разными парметрами
[myObject myMethodWithParam1:1 andParam2:nil];
[myObject myMethodWithParam1:0 andParam2:someObject];
[myObject myMethodWithParam1:1 andParam2:someObject];
...
```

# Методы

- Описываются селекторами на более низком уровне. Селектор - строковая сигнатура метода.
- Когда вызывается метод, на самом деле объекту посылается сообщение, содержащее селектор и параметры. (Отсюда - посылка сообщения нулевому объекту.)

```
// Посылка сообщения на Objective-C:
[myObject myMethodWithParam1:1 andParam2:
someObject];
// Транслируется в следующий C код:
// Создается селектор по строковой сигнатуре
SEL selector = sel_registerName("myMethodWithParam1:
andParam2:");
// Посылается сообщение объекту с этим селектором и
параметрами:
objc_msgSend(myObject, selector, 1, someObject);
// Далее внутри objc_msgSend ищется реализация
селектора в dispatch-таблице объекта вплоть до
последнего суперкласса.
```

# Методы класса

- Работают аналогично методам экземпляра, но на уровне всего класса.
- Внутри их реализации недоступны переменные и методы экземпляров.
- Можно воспринимать их как глобальные методы, но с логической привязкой к конкретному классу.

```
// Описание метода класса
+ (MyClass *)defaultObject;

// Реализация метода
+ (MyClass *)defaultObject {
MyClass *retVal = [[MyClass alloc] init];
retVal.myProperty = 1000;

_myIvar = 10; // Недоступно
return retVal;
}
```

# self

Указатель на текущий экземпляр класса:

```
self.myProperty = 20;
[self myMethodWithParam1:0 andParam2:nil];
```

Указатель на текущий класс (если обращаться из метода класса):

```
MyClass *foo = [self defaultObject];
```

# Свойства

- Синтаксический сахар вокруг переменной экземпляра (ivar) и геттера и сеттера.
- Геттер (от англ. get) - метод, возвращающий значение ivar.
- Сеттер (от англ. set) - метод, устанавливающий значение ivar.

```
// Код типа
@property float height;
// на самом деле генерирует (синтезирует):
// - Переменную экземпляра
float _height;
// - А также геттер:
- (float)height {
return _height;
}

// - И сеттер:
 - (void)setHeight:(float)height {
_height = height;
}
```

Поэтому следующие вызовы эквивалентны:

```
self.height = 0.0f;
[self setHeight:0.0f];
// а также
float someHeight = self.height;
someHeight = [self height];
```

Зачем это нужно?

- Для удобства разделения методов и свойств.
- Неявное управление памятью при объявлении свойства и выставлении его атрибутов.
- Можно переопределять геттеры и сеттеры.

# Жизненный цикл объекта

1. Выделяется память под объект.
2. Объект инициализируется.
3. Объект живет, пока на него кто-либо ссылается ("владеет" им).
4. На объект никто не ссылается - объект удаляется из памяти.

Все объекты в Cocoa и Cocoa Touch наследуются от NSObject.
Поддержка жизненного цикла у NSObject:
- Метод +(instancetype)alloc; - выделяет память для объекта. Нельзя переопределять.
- Метод -(instancetype)init; - инициализирует объект.
- Метод -(void)dealloc; - вызывается средой выполнения, когда количество ссылок на объект становится равным нулю. Вызывать напрямую запрещено.