Noveo
Software University

1. Objective-C

Noveo University — iOS

Александр Горбунов

Сегодня

- основы языка
- структура программы
- принципы ООП
- типичные паттерны

Objective-C

- объектно-ориентированный язык
- включает все возможности C
- богатый runtime
- язык поддерживается CLang и GCC
- платформонезависимый
- Objective-C++ (взаимодействие Objective-C и C++ кода с некоторыми ограничениями)

Структура программы

.h

header-файлы содержат **объявления** классов, протоколов, категорий, публичных методов и свойств, функций, констант, перечислений, макросов.

.m

implementation-файлы содержат **реализации** методов, функций, определения констант.

Структура программы

```
#include <LibraryName>
#include "FileName"
```

`include` работает как в C, на практике используется редко

```
#import <LibraryName>
#import "FileName"
```

`import` (в отличие от `include`) прощает многократные и циклические включения

```
@import Module
@import Module.SubModule
```

`@import` подключает *модуль*, импортируя заголовки и автоматически линкуя необходимые библиотеки

ООП в Objective-C

`@interface` — объявление **интерфейса** (класса)

```
@interface Number : NSObject
    ...
@end
```

`:` `FOO` — указание супер-класса. Можно делать свои корневые классы, но на практике всегда используется `NSObject`.

Множественного наследования нет.

Абстрактных классов нет.

ООП в Objective-C

`@protocol` — объявление **протокола**. Протокол содержит *список* методов без привязки к конкретным классам.

```
@protocol Printable <NSObject>
    ...
@end
```

`<FOO>` — указание родительского протокола. Можно делать свои корневые протоколы, но на практике всегда используется `NSObject`.

У методов не может быть реализации по умолчанию.

ООП в Objective-C

Объявление класса, реализующего протокол:

```
@interface Number : NSObject <Printable>
    ...
@end
```

Можно перечислять реализуемые протоколы через запятую.

ООП в Objective-C

`(FOO)` — объявление **категории** над классом:

```
@interface Number (Formatting)
    ...
@end
```

Категория дописывает методы к уже существующему классу.

Можно добавить методы, не прибегая к наследованию.

Категорий может быть несколько.

ООП в Objective-C

( ) — объявление **расширения** (анонимной категории) над классом:

```
@interface Number ()
    ...
@end
```

Обычно используется для объявления приватных свойств и методов в .m файле.

ООП в Objective-C

Секция реализации класса и его расширения:

```
@implementation Number
    ...
@end
```

Секция реализации категории класса:

```
@implementation Number (Printable)
    ...
@end
```

ООП в Objective-C

Объявление **экземпляров** класса:

```
#import "Number.h"

...
    Number *number_A;
    Number *number_B;
...
```

ООП в Objective-C (ещё раз)

Класс может иметь один родительский класс или не иметь вовсе.
Все классы конкретные (могут быть инстанциированы).

Интерфейс класса обозначается ключевым словом `interface`.

Реализация класса обозначается ключевым словом `implementation`.

```
@interface ClassName : SuperClassName
    объявление методов класса
@end


@implementation ClassName
    реализация методов класса
@end
```

ООП в Objective-C (ещё раз)

Протокол — список методов без привязки к конкретному классу. По применению соответствует интерфейсу Java или абстрактному классу C++. В объявлении протокола может быть указано, какие другие протоколы он включает в себя. Каждый класс может отвечать нескольким протоколам.

```
@protocol ProtocolName <SuperProtocol_A, SuperProtocol_B>
    объявление методов протокола
@end

@interface ClassName : SuperClass <ProtocolName, OtherProtocol>
    объявление методов класса
@end

@implementation ClassName
    реализация методов класса и его протоколов
@end
```

ООП в Objective-C (ещё раз)

Категория — список методов, которые мы дописываем к уже существующему классу.

```
@interface ClassName (CategoryName)
    объявление методов категории
@end

@implementation ClassName (CategoryName)
    реализация методов категории
@end
```

ООП в Objective-C (ещё раз)

Расширение — анонимная категория. Обычно используется для объявления приватных методов и свойств.

```
@interface ClassName ()
    объявление методов расширения
@end

@implementation ClassName
    реализация методов класса и расширения
@end
```

ООП в Objective-C (резюме)

- **Интерфейс** — класс. Может иметь только один родительский класс.
- **Протокол** — аналог абстрактного класса. Каждый класс может отвечать нескольким протоколам.
- **Категория** — список методов, которые мы дописываем к уже существующему классу без применения наследования. Каждый класс может иметь несколько категорий.
- **Расширение** — анонимная категория. Обычно используется для объявления приватных методов и свойств. У каждого класса может быть только одно расширение.

Методы в Objective-C

Метод объекта

- вызывается у объекта
- объявление начинается с "-"
- есть доступ к `self` и `super`
- есть доступ к полям и свойствам объекта
- может вызывать другие методы текущего объекта

Объявление метода:

```
@interface Number : NSObject
- (void)methodOfInstance;
@end
```

Методы в Objective-C

Метод класса

- вызывается у класса
- объявление начинается с "+"
- `self` — объект-класс
- может вызывать другие методы текущего класса
- аналог статического метода в C++
- в Objective-C нет настоящих статических методов

Объявление метода:

```
@interface Number : NSObject
+ (void)methodOfClass;
@end
```

Методы в Objective-C

Методы с параметрами:

```
@interface Number : NSObject
- (int)methodWithParameterA:(int)a;
- (int)methodWithParameterA:(int)a andB:(int)b;
@end
```

Именем метода является вся строка кроме параметров, но включая двоеточия:

```
methodWithParameterA:
methodWithParameterA:andB:
```

Методы в Objective-C

Реализация метода:

```
@implementation Number

- (int)methodWithParameterA:(int)a andB:(int)b
{
    return self.value * a * b;
}

@end
```

Методы в Objective-C

```
@interface Number : NSObject
- (int)methodWithParameterA:(int)a andB:(int)b;
- (void)method;
+ (int)classMethod;
@end
```

Вызов методов:

```
    Number *number;
    ...
    [number methodWithParameterA:8 andB:9];
    [number method];
    [Number classMethod];
```

~~Методы~~ сообщения в Objective-C

- Объекты отправляют друг другу **сообщения**.
- Для идентификации методов есть отдельный тип — SEL (селектор).
- Во время компиляции неизвестно, отвечает ли объект на селектор.
- Во время исполнения можно спросить у объекта, отвечает ли он на конкретный селектор.
- Сообщения отправляются объектам через указатель, а отправка сообщения в nil не вызывает никаких действий.

Сообщения в Objective-C

```
@interface ABC
- (void)actionWithA:(int)a andB:(int)b;
@end
```

... во время исполнения можно спросить у объекта, отвечает ли он на конкретный селектор:

```
    ...
    SEL action = @selector(actionWithA:andB:);
    if ([x respondsToSelector:action]) {
        ...
    }
    ...
```

Объекты в Objective-C

`void *p` — указатель на произвольный участок памяти

- нулевой указатель — `NULL`
- допустима адресная арифметика
- не поддерживается ARC (система автоматического управления памятью)

`id p` — указатель на произвольный Objective-C объект

- нулевой указатель — `nil`
- адресная арифметика запрещена
- поддерживается ARC

Объекты в Objective-C

`ClassName *p` — указатель на объект конкретного класса

`self` — указатель на текущий объект (у которого вызван метод), аналог `this`

`super` — указатель на текущий объект (у которого вызван метод), интерпретируемый как экземпляр супер-класса

`instancetype` — тип "указатель на объект этого класса" (которому принадлежит метод), допустим только как тип возвращаемого значения

Поля и свойства объекта

Поле (ivar, instance variable)

- переменная
- хранит состояние объекта
- может быть публичным или приватным

```
@interface Number : NSObject
{
    int myIntValue;
}
@end
```

Поля и свойства объекта

Поле может быть публичным или приватным:

```
@interface Number : NSObject
{
    @private
    int privateIvar_1;
    int privateIvar_2;

    @protected
    int protectedIvar_1;

    @public
    int publicIvar_1;
}
@end
```

Поля и свойства объекта

```
@interface Number : NSObject
{
    @public
    int publicIvar;
}
@end
```

Обращение к полю:

```
    Number *number = ...;
    number->publicIVar = 8;
```

Поля редко используются напрямую

Поля и свойства объекта

Свойство

- виртуальная переменная
- хранит состояние объекта
- доступно тем, кто видит объявление
- доступ к значению через методы доступа
- можно сделать потоко-безопасным
- можно явно декларировать способ управления памятью

```
@interface Number : NSObject
@property int value;
@end
```

Поля и свойства объекта

Атрибуты свойств

`copy` — копирование при присваивании. Работает только для объектов.

`strong` — сильная ссылка (увеличивает счётчик ссылок). Объект не будет уничтожен, пока на него есть сильные ссылки. Работает только для объектов.

`weak` — слабая ссылка (не увеличивает счётчик ссылок). Если объект будет уничтожен, все слабые ссылки автоматически обнулятся. Работает только для объектов.

`assign` — сырое значение или слабая ссылка (но не обнуляется автоматически). Работает и для скалярных типов и для объектов.

Поля и свойства объекта

Атрибуты свойств

`readonly` — только для чтения (в случае со скалярами — они неизменны, в случае с объектами — неизменен только указатель).

`readwrite` — для чтения и записи.

Поля и свойства объекта

Атрибуты свойств

`nonatomic` — обычный неатомарный доступ.

`atomic` — атомарный доступ (по смыслу аналогично транзакции).

Поля и свойства объекта

Атрибуты свойств

```
@interface Number : NSObject
@property int a;
@property (atomic) int b;
@property (nonatomic, readonly) int c;
@property (nonatomic, copy) NSString *s;
@end
```

Поля и свойства объекта

```
@implementation Number
@synthesize value = _value;
...
@end
```

превращается в

```
- (int)value {
    return _value;
}

- (void)setValue:(int)value {
    _value = value;
}
```

Поля и свойства объекта

```
@implementation Number
@synthesize value = _value;
...
@end
```

генерируется автоматически

Поля и свойства объекта

Свойства

- Меньше проблем с областью видимости и наследованием
- Могут быть атомарным или нет (`atomic/nonatomic`)
- Могут быть защищены от записи или нет (`readonly/readwrite`)
- Можно добавить побочные эффекты для изменения значения
- Можно вычислять значение при чтении
- Позволяют использовать dot-нотацию
- Позволяют доступаться до значения свойства в runtime по имени свойства (KVC)

NSObject

Базовый класс почти для всего, с чем мы имеем дело.

```
+ alloc
– init
– copy
– mutableCopy
– dealloc
+ new
+ class
+ superclass
+ isSubclassOfClass:
...
```

NSObject

Протокол, которому отвечает почти всё, с чем мы имеем дело.

```
– class
– isEqual:
– hash
– isKindOfClass:
– respondsToSelector:
– conformsToProtocol:
– description
– performSelector:
...
```

NSObject и интроспекция

Во время выполнения программы мы можем спросить у любого объекта, отвечает ли он на нужный селектор (сообщение).

```
NSArray *array;
for (id object in array) {
    if ([object respondsToSelector:@selector(draw)]) {
        [object performSelector:@selector(draw)];
    }
}
```

NSObject и интроспекция

Во время выполнения программы мы можем спросить у любого объекта, реализует ли он нужный протокол.

```
NSArray *array;
for (id object in array) {
    if ([object conformsToProtocol:@protocol(Drawable)]) {
        id<Drawable> drawable = object;
        [drawable draw];
    }
}
```

NSObject и интроспекция

Во время выполнения программы мы можем спросить у любого объекта, является ли он экземпляром класса.

```
NSArray *array;
for (id object in array) {
    if ([object isKindOfClass:[Circle class]]) {
        [object performSelector:@selector(draw)];
    }
}
```