

Разработка iOS приложений

Лекция 2.

Основы языка Objective-C

Objective-C

Objective-C, известный также как Objective C, ObjC или Obj-C — компилируемый объектно-ориентированный язык программирования корпорации Apple, построенный на основе языка Си и парадигм Smalltalk. В частности, объектная модель построена в стиле Smalltalk — то есть объектам посылаются сообщения.

Язык Objective-C является надмножеством языка Си, поэтому Си-код полностью понятен компилятору Objective-C.

Кроме того, при соблюдении определенных правил, можно также использовать и C++.

Objective-C: Объявление класса



RssItem.h Заголовочный файл для класса. Он объявляет открытое API класса

Objective-C: Объявление класса



RssItem.h Заголовочный файл для класса. Он объявляет открытое API класса

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

Мы должны импортировать заголовочный файл с описанием класса-предка

```
@interface RssItem : NSObject
```

```
@end
```

Objective-C: Объявление класса



`RssItem.h` Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
```

Имя класса

```
@end
```

Objective-C: Объявление класса



RssItem.h Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
```

Имя класса предка

```
@end
```

Objective-C: Объявление класса



RssItem.h Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
```

```
{  
  
}  
}
```

Переменные класса
объявляются тут

```
@end
```

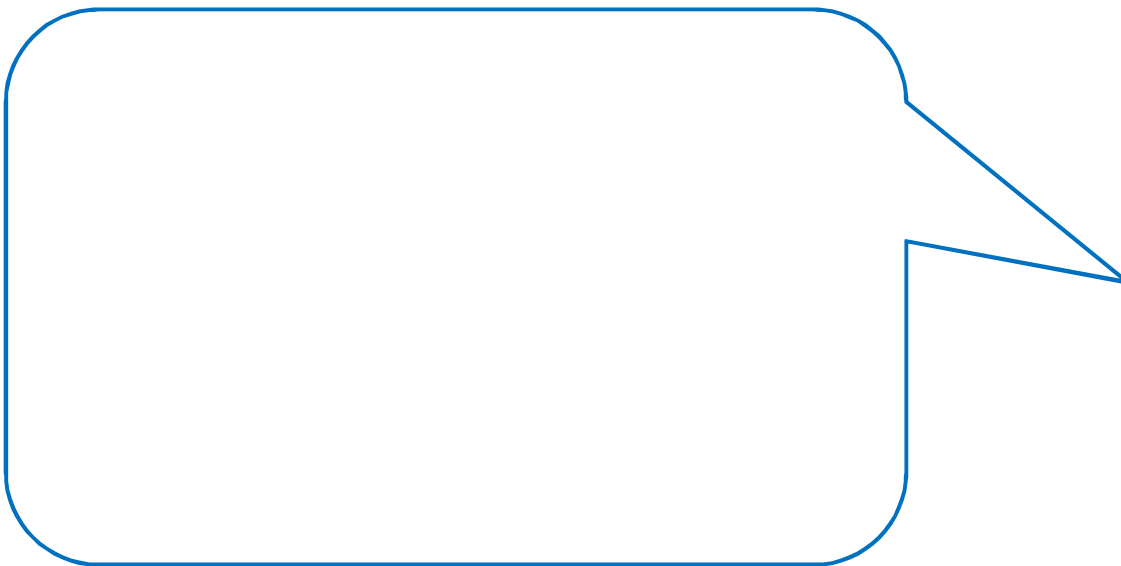
Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject  
{  
    NSString* title;  
    NSString* guid;  
}
```



```
@end
```

Методы класса
объявляются тут

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject  
{  
    NSString* title;  
    NSString* guid;  
}
```

```
-( void ) setTitle: ( NSString* )title;  
-( NSString* ) saveImage: ( UIImage* )image;
```

```
@end
```

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
{
    NSString* title;
    NSString* guid;
}
```

У метода нет возвращаемого значения

```
-( void ) setTitle: ( NSString* )title;
-( NSString* ) saveImage: ( UIImage* )image;
```

```
@end
```

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
{
    NSString* title;
    NSString* guid;
}
```

Название метода – setTitle:

```
-( void ) setTitle: ( NSString* )title;
-( NSString* ) saveImage: ( UIImage* )image;
```

```
@end
```

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
{
    NSString* title;
    NSString* guid;
}
```

Он принимает один параметр *строку* под названием title

```
-( void ) setTitle: ( NSString* )title;
-( NSString* ) saveImage: ( UIImage* )image;
```

```
@end
```

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
{
    NSString* title;
    NSString* guid;
}
```

И не забудьте про точку с запятой

```
-( void ) setTitle: ( NSString* )title;
-( NSString* ) saveImage: ( UIImage* )image;
```

```
@end
```

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
{
    NSString* title;
    NSString* guid;
}
```

```
-( void ) setTitle: ( NSString* )title;
-( NSString* ) saveImage: ( UIImage* )image;
```

Этот метод возвращает строку

```
@end
```

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject  
{  
    NSString* title;  
    NSString* guid;  
}
```

```
-( void ) setTitle: ( NSString* )title;  
-( NSString* ) saveImage: ( UIImage* )image;
```

И в качестве параметра принимает указатель на картинку.
Да здесь передается объект.

```
@end
```

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
{
    NSString* title;
    NSString* guid;
}
```

```
-( void ) setTitle: ( NSString* )title;
-( NSString* ) saveImage: ( UIImage* )image;
-( NSArray* ) categsByName: ( NSString* )name orType: ( id )type;
```

```
@end
```


Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
{
    NSString* title;
    NSString* guid;
}
```

```
-( void ) setTitle: ( NSString* )title;
-( NSString* ) saveImage: ( UIImage* )image;
-( NSArray* ) categsByName: ( NSString* )name orType: ( id ) type;
```

Этот метод принимает два параметра и называется
“categsByName:orType:”

```
@end
```

Objective-C: Объявление класса



[RssItem.h](#) Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject
{
    NSString* title;
    NSString* guid;
}
```

```
-( void ) setTitle: ( NSString* )title;
-( NSString* ) saveImage: ( UIImage* )image;
-( NSArray* ) categsByName: ( NSString* )name orType: ( id )type;
```

Он возвращает указатель на объект типа NSArray*
это неизменяемый массив из NSFoundation

```
@end
```

Objective-C: Объявление класса



RssItem.h Заголовочный файл для класса. Он объявляет открытое API класса

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

```
@interface RssItem : NSObject  
{  
    NSString* title;  
    NSString* guid;  
}
```

```
-( void ) setTitle: ( NSString* )title;  
-( NSString* ) saveImage: ( UIImage* )image;  
-( NSArray* ) categsByName: ( NSString* )name orType: ( id ) type;
```

Второй параметр имеет тип id, в Objective-C это означает «указатель на любой тип объекта»

```
@end
```

Objective-C: Определение класса



RssItem.m Файл определения. Здесь определяются и публичные и закрытые методы и переменные

```
#import "RssItem.h"
```

```
@implementation RssItem
```

```
@end
```

Objective-C: Определение класса



RssItem.m Файл определения. Здесь определяются и публичные и закрытые методы и переменные

```
#import "RssItem.h"  
  
@implementation RssItem
```

Мы должны загрузить наш заголовочный файл класса

```
@end
```

Objective-C: Определение класса



RssItem.m Файл определения. Здесь определяются и публичные и закрытые методы и переменные

```
#import "RssItem.h"
```

```
@implementation RssItem
```

```
-( void ) setTitle: ( NSString* )title  
{  
    <здесь реализация метода>  
}
```

Точка с запятой здесь
не нужна



```
@end
```

Objective-C: Определение класса



RssItem.m Файл определения. Здесь определяются и публичные и закрытые методы и переменные

```
#import "RssItem.h"
```

```
@implementation RssItem
```

```
-( void ) setTitle: ( NSString* )title  
{  
    self.title = title;  
}
```

```
-( NSString* ) saveImage: ( UIImage* )image  
{  
    NSString* file = @"image.png";  
    [ UIImagePNGRepresentation( image ) saveToFile: file ];  
    return file;  
}
```

```
@end
```

Objective-C: Определение класса



RssItem.m Файл определения. Здесь определяются и публичные и закрытые методы и переменные

```
#import "RssItem.h"
```

```
@implementation RssItem
```

```
-( void ) setTitle: ( NSString* )title  
{  
    self.title = title;  
}
```

```
-( NSString* ) saveImage: ( UIImage* )image  
{
```

```
    NSString* file = @"image.png";
```

```
    [ UIImagePNGRepresentation( image ) saveToFile: file ];  
    return file;
```

```
}
```

```
@end
```

Квадратный скобки обозначают посылку сообщений

Objective-C: Определение класса



RssItem.m Файл определения. Здесь определяются и публичные и закрытые методы и переменные

```
#import "RssItem.h"
```

```
@implementation RssItem
```

```
-( void ) setTitle: ( NSString* )title  
{  
    self.title = title;  
}
```

```
-( NSString* ) saveImage: ( UIImage* )image  
{  
    NSString* file = @"image.png";  
    [ UIImagePNGRepresentation( image ) saveToFile: file ];  
    return file;  
}
```

```
@end
```

Потом собственно само сообщение, которое необходимо послать

Objective-C: Определение класса



RssItem.m Файл определения. Здесь определяются и публичные и закрытые методы и переменные

```
#import "RssItem.h"
```

```
@implementation RssItem
```

```
-( void ) setTitle: ( NSString* )title  
{  
    self.title = title;  
}
```

```
-( NSString* ) saveImage: ( UIImage* )image  
{  
    NSString* file = @"image.png";  
    [ UIImagePNGRepresentation( image ) saveToFile: file ];  
    return file;  
}
```

```
@end
```

Указываются аргументы (в данном случае один) сообщения

Objective-C: Синтаксис метода

```
+( RssItem* ) rssItemWithTitle: ( NSString* )title;

-( void ) setTitle: ( NSString* )title;

-( NSArray* ) categsByName: ( NSString* )name orType: ( id )type;
```

Objective-C: Методы экземпляра класса

- Начинаются с дефиса

```
-( void ) setTitle: ( NSString* )title;
```

- Это те самые методы, которые Вы будете использовать
- Могут получить доступ к переменным экземпляра, как будто это локальные переменные
- Могут посылать сообщения self и super
- Пример вызова:

```
BOOL destroyed = [ plane dropBombAt: dropPoint ];
```

Objective-C: Методы класса

- Начинаются с плюса. Используются для выделения памяти, Singleton'ов, вспомогательных функций

```
+( RssItem* ) rssItemWithTitle: ( NSString* )title;
```

- Не могут получать доступ к переменным экземпляра класса
- Посылка методов self и super означает только посылку методов классу, но не экземплярам

- Пример вызова:

```
RssItem* item = [ RssItem rssItemWithTitle: @"a title" ];
```

Objective-C: Переменные экземпляра класса

- Область видимости

По умолчанию переменные экземпляра являются `@protected` (доступны только классу и его подклассам). Кроме того, могут быть объявлены как `@private` (доступны только классу) и `@public` (доступны всем).

- Синтаксис:

```
@interface MyObject : NSObject
{
    int foo;
    @private
    int eye;
    @protected
    int bar;
    @public
    int forum, apology;
    @private
    int jet;
}
```

Objective-C: Переменные экземпляра класса

- Область видимости

По умолчанию переменные экземпляра являются `@protected` (доступны только классу и его подклассам). Кроме того, могут быть объявлены как `@private` (доступны только классу) и `@public` (доступны всем).

- Синтаксис:

```
@interface MyObject : NSObject
{
    int foo;
    @private
    int eye;
    @protected
    int bar;
    @public
    int forum, apology;
    @private
    int jet;
}
```

protected

private

public

Objective-C: Создание объектов

- Чтобы создать объект в Objective-C требуется сделать два шага
 - Выделить память (делается с помощью метода `alloc` из `NSObject`)
 - Проинициализировать объект (делается с помощью методов `init...`)
- `alloc` выделяет память в куче для класса
- Каждый класс должен содержать как минимум один метод `init`
 - ✓ Но любой из них должен начинаться с «`init`» (соглашение)
 - ✓ Каждый класс потомок должен вызывать метод `init` родителя
 - ✓ В инициализатор вы должны передавать только важные переменные
 - ✓ Можно делать много разных инициализаторов, но все они должны вызывать какой то один определенный
 - ✓ Все инициализаторы должны возвращать тип `id`

Objective-C: Создание объектов

- Пример

```
@implementation MyObject

-( id ) init
{
    if ( self = [ super init ] )
    {
        // инициализируем подкласс тут
    }

    return self;
}

@end
```

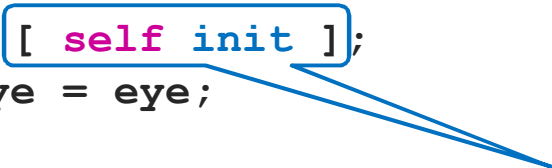
Objective-C: Создание объектов

- Дополнительный инициализатор

```
@implementation MyObject
```

```
-( id ) initWithEye: ( int )eye  
{  
    self = [ self init ];  
    self.eye = eye;  
  
    return self;  
}
```

```
@end
```



Мы обязаны вызвать
главный инициализатор

Objective-C: Получение объектов

- Однако создание – это не единственный метод получения объектов. Их Вам могут вернуть другие классы, если Вы попросите:

```
NSString* newDisplay = [ Display newDisplay ];  
NSArray* keys = [ dictionary allKeys ];  
NSString* lowerString = [ string lowercaseString ];  
NSNumber* n = [ NSNumber numberWithFloat: 42.0f ];  
NSDate* date = [ NSDate date ];
```

- Внимание вопрос: кто освобождает память для этих объектов?
- Сборки мусора на iOS нет (и не будет)
- Так как?

Objective-C: Получение объектов

- Однако создание – это не единственный метод получения объектов. Их Вам могут вернуть другие классы, если Вы попросите:

```
NSString* newDisplay = [ Display newDisplay ];  
NSArray* keys = [ dictionary allKeys ];  
NSString* lowerString = [ string lowercaseString ];  
NSNumber* n = [ NSNumber numberWithFloat: 42.0f ];  
NSDate* date = [ NSDate date ];
```

- Внимание вопрос: кто освобождает память для этих объектов?
- Сборки мусора на iOS нет (и не будет, ARC не в счет – это не сборка мусора)
- Так как?
- Ответ – подсчет ссылок!

Objective-C: Подсчет ссылок

- Как это работает?
Простой набор правил, которым надо следовать
- Вы берете на себя владение объектом, если хотите сохранить на него указатель
Несколько владельцев для одного объекта – это нормально (и довольно частая ситуация)
- Когда Вы закончили работать с объектом, необходимо прекратить владение
- Когда все отказались от объекта – он удаляется
После этого, Ваша программа упадет, если Вы вдруг отправите сообщение этому объекту

Objective-C: Владение объектом

- Когда вы берете на себя владение объектом?
Когда вызываете функции `new`, `alloc`, `copy`, ...
Вы можете завладеть объектом, который вы не создавали, отправив ему сообщение `retain`
- Так кто же владеет объектом, который вы не создавали?
Вы можете им пользоваться в рамках функции, где Вы его получили, но Вы им не владеете

Вы можете из завладеть с помощью сообщения `retain`
- Как прекратить владение объектом?
Очень просто – достаточно отправить ему сообщение `release`.
Никогда не посылайте `release` объектам, которыми Вы не владеете

Objective-C: Владение объектом


- Когда вы берете на себя владение объектом?
Когда вызываете функции `new`, `alloc`, `copy`, ...
Вы можете завладеть объектом, который вы не создавали, отправив ему сообщение `retain`
- Так кто же владеет объектом, который вы не создавали?
Вы можете им пользоваться в рамках функции, где Вы его получили, но Вы им не владеете

Вы можете из завладеть с помощью сообщения `retain`
- Как прекратить владение объектом?
Очень просто – достаточно отправить ему сообщение `release`.
Никогда не посылайте `release` объектам, которыми Вы не владеете

Objective-C: Временное владение

- Иногда возникают ситуации, когда владение объектом истекает где-то в будущем
- Лучше на примере:

```
- ( Money* ) showMeTheMoney: ( double ) amount {  
    Money* money = [ [ Money alloc ] init: amount ];  
    return money;  
}
```

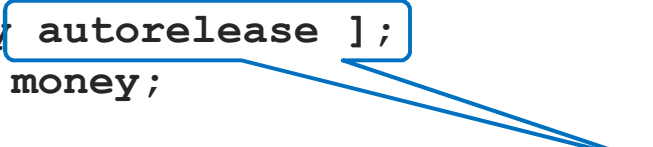


Мы вернули указатель, но не выполнили требование сделать release.
Потому что если мы это сделаем, мы вернем уничтоженный объект.

Objective-C: Временное владение

- Иногда возникают ситуации, когда владение объектом истекает где-то в будущем
- Лучше на примере:

```
- ( Money* ) showMeTheMoney: ( double ) amount {  
    Money* money = [ [ Money alloc ] init: amount ];  
    [ money autorelease ];  
    return money;  
}
```



Мы выполнили требование вызывать release, но дали шанс пользователям сделать retain или сору где-то в дальнейшем

```
Money* myMoney = [ bank showMeTheMoney: 4500.0f ];  
[ myMoney retain ];
```

Objective-C: Другие правила владения

- Коллекции берут на себя владение объектом, когда мы добавляем последний NSArray, NSSet, NSDictionary (NSDictionary владеет и ключем и значением)
При удалении из коллекции владение прекращается
- Думайте о строках @"string" как autorelease
- Для строк NSString лучше посылать copy, нежели retain
Это позволяет сохранять их неизменяемыми
Это не освобождает Вас от необходимости вызывать release
- Следует вызывать release как можно раньше
Лучше всего сразу, как только Вы закончили пользоваться объектом

Objective-C: Деаллокация

- Что происходит с объектом, когда вызывается последний release?
Вызывается специальный метод dealloc объекта
Память возвращается в кучу
- Вы должны переопределить метод dealloc своего класса, но **НИКОГДА** не вызывать его!
Он вызывается только после вызова последнего release автоматически
Единственное исключение – необходимо вызывать [super dealloc] из вашего метода
- Пример

```
- ( void ) dealloc
{
    [ brain release ];
    [ otherVariables release ];
    [ super dealloc ];
}
```

Objective-C: Свойства

- Забудьте все что вы видели на предыдущих слайдах!

Пометьте все ваши свойства как `@private`

И используйте `@property` для доступа к ним

- Создайте методы для доступа к внутренним переменным:

```
@interface MyObject : NSObject
```

```
{
```

```
  @private
```

```
    int eye;
```

```
}
```

```
-( int ) eye;
```

```
-( void ) setEye: ( int ) anInt;
```

```
@end
```

- Теперь кто угодно сможет получить доступ к вашим переменным с помощью «точечной нотации» (dot notation)

```
someObject.eye = newEyeValue; // установить (setEye:)
```

```
int eyeValue = someObject.eye; // получить (eye)
```

Objective-C: Свойства

- Забудьте все что вы видели на предыдущем слайде!

Пометьте все ваши свойства как `@private`

И используйте `@property` для доступа к ним

- Создайте методы для доступа к внутренним переменным:

```
@interface MyObject : NSObject
{
    @private
    int eye;
}

- ( int ) eye;
- ( void ) setEye: ( int ) anInt;

@end
```

ВАЖНО! Переменные экземпляра всегда должны начинаться с маленькой буквы. В то время как после set всегда должна идти заглавная буква, иначе точечная нотация работать не будет!

- Теперь кто угодно сможет получить доступ к вашим переменным с помощью «точечной нотации» (dot notation)

```
someObject.eye = newEyeValue; // установить (setEye:)
int eyeValue = someObject.eye; // получить (eye)
```

Objective-C: Свойства

- `@property`

Вы можете заставить компилятор генерировать методы get/set для Вас

```
@interface MyObject : NSObject
{
    @private
        int eye;
}
```

```
@property int eye;
```

```
@end
```

- Если вы используете спецификатор `readonly`, то будет сгенерирован только метод `get`

```
@property ( readonly ) int eye;
```

Objective-C: Свойства

- `@property` не обязательно должно называться также как переменная экземпляра

```
@interface MyObject : NSObject
{
    @private
        int p_eye;
}

@property int eye;

@end
```

- Переменной экземпляра может не быть вообще

```
@interface MyObject : NSObject
{
}

@property int eye;

@end;
```

Objective-C: Свойства

- Но как бы Вы не объявляли, Вы должны реализовать

```
@interface MyObject : NSObject
{
    @private
        int eye;
}

@property int eye;

@end

@implementation MyObject

- ( int ) eye {
    return eye;
}

- ( void ) setEye: ( int )anInt {
    eye = anInt;
}

@end;
```


Objective-C: Свойства

- Или в случае когда переменная экземпляра называется по другому

```
@interface MyObject : NSObject
{
    @private
        int p_eye;
}

@property int eye;

@end

@implementation MyObject

- ( int ) eye {
    return p_eye;
}

- ( void ) setEye: ( int ) anInt {
    p_eye = anInt;
}

@end;
```

Objective-C: Свойства

- А в случае без переменной экземпляра

```
@interface MyObject : NSObject
{
}

@property ( readonly ) int eye;

@end

@implementation MyObject

-( int ) eye {
    <какой-нибудь алгоритм для расчета eye>;
    return <вычисленное значение для eye>;
}

@end;
```

Objective-C: Свойства

- Вы можете попросить компилятор выполнить реализацию за Вас с помощью `@synthesize`

```
@interface MyObject : NSObject
{
    @private
        int eye;
}

@property ( readonly ) int eye;

@end

@implementation MyObject

@synthesize eye;

@end;
```

Objective-C: Свойства

- А если переменная экземпляра называется по другому, то уточнить

```
@interface MyObject : NSObject
{
    @private
        int p_eye;
}

@property ( readonly ) int eye;

@end

@implementation MyObject

@synthesize eye = p_eye;

@end;
```

Objective-C: Свойства

- Хотя даже если Вы пользуетесь @synthesize, Вы все равно можете написать свою реализацию

```
@implementation MyObject

@synthesize eye;

-( void ) setEye: ( int )anInt {
    if ( anInt > 0 ) eye = anInt;
}

@end;
```

Метод -(int)eye будет реализован для Вас @synthesize
В качестве setEye будет использоваться Ваша реализация
Если Вы реализуете и сеттер и геттер, то @synthesize будет проигнорировано

Objective-C: Свойства. Управление памятью

- Есть три вида спецификатора свойств

assign – просто присваивает (подходит для простых типов – int, double, float, bool, ...)

retain – удаляет старый объект (release) и для нового объекта получает владение (retain)

copy – удаляет старый объект (release) и для нового получает копию (copy)

```
@property (retain) NSArray* arrayOfObjects;  
@property (copy) NSString* someString;  
@property (assign) int value;
```

Objective-C: Основные классы NSFoundation

- NSObject
- NSString
- NSNumber
- NSArray
- NSMutableArray
- NSDictionary
- NSMutableDictionary

Objective-C: NSObject

- alloc
- init
- retain
- release
- dealloc
- ...

Objective-C: NSString

```
NSString* string = @"static string";
```

```
NSString* formatString = [ NSString stringWithFormat: @"%d - %@",  
count, description ];
```

Objective-C: NSNumber

```
NSNumber* num = [ NSNumber numberWithInt: 5 ];
```

```
NSInteger int = [ num intValue ];
```

Objective-C: NSArray

```
NSArray* array = [ NSArray array ]; // alloc, init, autorelease!
```

```
NSUInteger count = [ array count ];
```

```
RssItem* item = ( RssItem* )[ items objectAtIndex: 1 ];
```

```
for ( RssItem* item in items )  
{  
    ...  
}
```

Objective-C: NSMutableArray

```
NSMutableArray* itemList = [ NSMutableArray array ];  
  
[ itemList addObject: item ]; // retain!  
  
[ itemList removeObject: item ]; // release!
```

Objective-C: NSDictionary

```
NSDictionary* dict = [ item someDictionary ];

NSNumber* count = [ dict objectForKey: @"fruits" ];

if ( ! [ [ dict allKeys ] contains: @"testkey" ] ) ...

for ( NSString* key in dict )
{
    id value = [ dict objectForKey: key ];
}
```

Objective-C: NSMutableDictionary

```
NSMutableDictionary* dict = [ NSMutableDictionary dictionary ];  
  
[ dict setObject: obj forKey: @"key" ];  
  
[ dict removeObjectForKey: [ NSNumber numberWithInt: 5 ] ];
```

Практическое задание

В ходе курса мы разработаем одно простое с виду приложение – «Читалка RSS»

В качестве первого занятия требуется сделать следующее:

- Реализовать класс RSS
- Реализовать класс RSSChannel
- Реализовать класс RSSItem
- RSS должен возвращать predetermined в коде (hard coded) канал RSSChannel
- RSSChannel должен возвращать predetermined в коде набор элементов (RSSItem's)

Пример RSS можно взять например тут: <http://itw66.ru/rss/index>