Noveo
Software University

4. Foundation framework

Noveo University — iOS

Константин Мамаев

План лекции

- Что такое Foundation?
- Строки (NSString)
- Коллекции (NSArray, NSDictionary, NSSet)
- Обертки (NSData, NSValue, NSNumber)
- Дата и время (NSDate)
- Ошибки (NSError, NSException)
- Бонус

Foundation - это ...

- Фреймворк, созданный Apple, который доступен как iOS, так и OS X разработчикам.
- Фреймворк играет роль «базы» Objective-C классов и позволяет в некоторой мере абстрагироваться от конкретной ОС.

Справочная литература

- Документация Apple
- http://nshipster.com
- http://rypress.com

NSString + NSMutableString

C string vs NSString

C string

```
const char *cString = "Hello world";
```

NSString

```
NSString *objCString = @"Hello world";
```

- stringWithFormat:

```
int year = 2014;
NSString *message = [NSString stringWithFormat:@"Today is year %d", year];
// Today is year 2014

NSString *anotherMessage =
    [NSString stringWithFormat:@"Previous message was '%@'", message];
// Previous message was 'Today is year 2014'
```

это, наверное, самый часто используемый метод у строк

Format specifiers

Точно такие же, как у printf + `%@` для объектов

Основные:

- `%d`, `%i` - для int
- `%f` - для float
- `%s` - для строки C

NSRange

```
typedef struct _NSRange {
    NSUInteger location;
    NSUInteger length;
} NSRange;
```

NSRange используется в методах, которые что-то ищут или заменяют. Если ничего найти не удалось, они возвращают **NSNotFound** в *location*

Поиск подстроки

```
NSRange range = [@"Porsche Carrera" rangeOfString: @"Car"];
// {.location = 8, .length = 3}
```

… и еще пара методов

```
int length = [@"Hello world" length];
// 11

unichar ch = [@"Abcdef" characterAtIndex:3];
NSLog(@"%c", ch); // d
```

но они довольно редко используются, особенно второй

Создание новых строк на основе имеющихся

```
NSString *name = @"Steve";
NSString *surname = @"Jobs";
NSString *space = @" ";
NSString *fullName = [[name stringByAppendingString:space]
    stringByAppendingString:surname];
// Steve Jobs
NSString *king = @"King";
NSString *otherFullName = [[fullName stringByReplacingOccurrencesOfString:surname
    withString:king] stringByReplacingOccurrencesOfString:space withString:@"n "];
// Steven King
```

методы, создающие новую строку из имеющейся, обычно начинаются со stringBy...

Получение подстроки

```
NSString *car = @"Porsche Carrera";
NSString *model = [car substringFromIndex:8]; //Carrera
```

Смена регистра

```
NSString *st = @"sTRinG";
NSString *lower = [st lowercaseString]; // string
NSString *upper = [st uppercaseString]; // STRING
NSString *capital = [st capitalizedString]; // String
```

NSMutableString

Строка **NSString** неизменяема.

Однако, у **NSString** есть подкласс **NSMutableString** - для изменяемых строк.

Дополнительные методы

```
- (void)insertString:(NSString *)aString atIndex:(NSUInteger)loc;
- (void)deleteCharactersInRange:(NSRange)range;
- (void)appendString:(NSString *)aString;
- (void)appendFormat:(NSString *)format, ... NS_FORMAT_FUNCTION(1,2);
- (void)setString:(NSString *)aString;
```

Пример

```
NSMutableString *st = @"Mutable string";
[st appendString:@" ****"];
```

Что произойдёт?

*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: 'Attempt to mutate immutable object with appendString:'

Правильная реализация

```
NSMutableString *st = [@"Mutable string" mutableCopy];
```

или так:

```
NSString *st = [[NSMutableString alloc] initWithString:@"Mutable string"];
```

Задание для самостоятельного изучения

- Прочитать документацию к NSString.

Коллекции

Коллекции — это ...

Своеобразные хранилища для разнородных элементов.

- Основные: *NSArray*, *NSDictionary*, *NSSet*
- Экзотические: *NSCountedSet*, *NSOrderedSet*, *NSIndexSet*, *NSCharacterSet*
- Могут быть изменяемые и неизменяемые (mutable и immutable)
- Могут быть типизированные и нетипизированные *(NSArray < NSString *>, NSDictionary< Car *>, NSSet< UIView *>)*

Коллекции «на каждый день»

- NSArray - массив (элементы хранятся по порядку)
- NSDictionary - словарь (хранит пары ключ:значение; ключи не могут повторяться)
- NSSet - множество (хранит элементы без учета порядка)

Mutable и Immutable

NSArray, NSDictionary, NSSet - неизменяемые. После их создания ни добавить, ни удалить элементы нельзя.

NSMutableArray, NSMutableDictionary, NSMutableSet - изменяемые.

Mutable коллекции

Наследуют методы immutable коллекций + добавляют свои

Нельзя изменять коллекцию, пока ее перебираешь!

Пример

```
NSMutableArray *cities =
    [NSMutableArray arrayWithArray:@[@"Moscow", @"St. Petersburg", @"Kiev"]];
[cities enumerateObjectsUsingBlock:^(NSString *obj, NSUInteger idx, BOOL *stop) {
        if ([obj length] > 4) {
            [cities removeObject:obj];
        }
    }];
```

***** Terminating app due to uncaught exception 'NSGenericException', reason: '*** Collection <__NSArrayM: 0x100401c80> was mutated while being enumerated.'**

Правильная реализация

```
NSMutableArray *cities =
    [NSMutableArray arrayWithArray:@[@"Moscow", @"St. Petersburg", @"Kiev"]];
NSArray *copy = [cities copy];
[copy enumerateObjectsUsingBlock:^(NSString *obj, NSUInteger idx, BOOL *stop) {
        if ([obj length] > 4) {
            [cities removeObject:obj];
        }
    }];
```

Типизированные коллекции (Lightweight Generics)

```
NSMutableArray<NSString *> *array = [NSMutableArray array];
[array addObject:@5];
```

Вызовет предупреждение компилятора, но программа будет успешно выполнена!

```
NSDictionary<NSString *, NSNumber *> *dictionary = @{@"five": @5};
NSSet<NSString *> *set = [NSSet setWithArray:@[@"one", @"two", @"three"]];
```

Не будет хуже, если все коллекции в вашей программе будут типизированными

Как поместить в коллекцию нечто, что не является Objective-C объектом?

- число (*int, float, BOOL, NSInteger, …*) - используем NSNumber
- C struct - NSValue
- бинарные данные (*void **) - NSData
- отсутствие данных (*nil*) - NSNull

NSNull

```
[NSNull null]; // синглтон.
```

На равенство можно проверять с помощью **==**, но все же лучше использовать *-isEqual:*.

Пример

```objc
// данные, которые мы хотим поместить в массив
int x = 10;
NSRange range = NSMakeRange(10, 3);
void *pointer = (void *)&range.length;

// обертки для этих данных
NSNumber *xObj = @(x);
NSValue *rangeObj = [NSValue valueWithRange:range];
NSData *pointerObj = [NSData dataWithBytes:pointer length:sizeof(range.length)];

// создание массива
NSArray *array = @[xObj, rangeObj, pointerObj, [NSNull null]];

// получаем данные обратно
NSValue *value = (NSValue *)array[1];
NSLog(@"%@", value); // NSRange: {10, 3}
```

Проверки на равенство

```
obj1 == obj2
```

проверка, что **obj1** и **obj2** ссылаются на один и тот же объект (**не то, что нам нужно**).

```
[obj1 isEqual:obj2];
```

проверка, что **obj1** и **obj2** идентичны (**то, что нужно**).

Проверки на равенство

```
- (BOOL)isEqualToArray:(NSArray *)otherArray;
- (BOOL)isEqualToDictionary:(NSDictionary *)otherDictionary;
- (BOOL)isEqualToSet:(NSSet *)otherSet;
```

NSArray + NSMutableArray

NSArray — это ...

- Упорядоченный набор объектов.
- Нумерация начинается с 0, объекты могут быть любого класса.
- Может быть типизированный и нетипизированный.

Создание массивов

Создание нетипизированного массива.

```
NSArray *array = @[@1, @"Objective-C"];
NSArray *array2 = [NSArray arrayWithArray:array];
NSArray *array3 = [[NSArray alloc] initWithArray:array];
```

Создание типизированного массива.

```
    NSArray<NSString *> *array = @[@1, @"Objective-C"];
    NSArray<NSObject *> *array2 = [NSArray arrayWithArray:array];
    NSArray<NSString *> *array3 = [[NSArray alloc] initWithArray:array];
```

Обычно методам объекта (*-initWithArray:*) соответствует метод класса (*+arrayWithArray:*) с похожим названием.

Как создать новый массив на основе имеющегося?

- Дополнить имеющийся массив
- Получить подмассив из имеющегося массива
- Сортировать имеющийся массив

Дополнение массива

```
- (NSArray *)arrayByAddingObject:(id)anObject;
- (NSArray *)arrayByAddingObjectsFromArray:(NSArray *)otherArray;
```

Взятие подмассива

```
- (NSArray *)subarrayWithRange:(NSRange)range;
- (NSArray *)filteredArrayUsingPredicate:(NSPredicate *)predicate;
```

Сортировка массива

```
- (NSArray *)sortedArrayUsingComparator:(NSComparator)cmptr;
- (NSArray *)sortedArrayWithOptions:(NSSortOptions)opts
   usingComparator:(NSComparator)cmpt;
```

```
typedef NSComparisonResult (^NSComparator)(id obj1, id obj2);
```

```
typedef NS_OPTIONS(NSUInteger, NSSortOptions) {
    NSSortConcurrent = (1UL << 0),
    NSSortStable = (1UL << 4),
};
```

```
typedef NS_ENUM(NSInteger, NSComparisonResult) {
    NSOrderedAscending = -1L,
    NSOrderedSame,
    NSOrderedDescending
};
```

Определяем размер коллекции

```
- (NSUInteger)count;
```

Порядковый номер объекта внутри массива

```
- (NSUInteger)indexOfObject:(id)anObject;
- (NSUInteger)indexOfObjectIdenticalTo:(id)anObject;
```

Индексы элементов массива, удовлетворяющих заданым условиям

```
- (NSIndexSet *)indexesOfObjectsWithOptions:(NSEnumerationOptions)opts
    passingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL *stop))predicate;
```

+ еще несколько методов

Чтение массива с «диска» и запись массива на «диск»

```
- (id)initWithContentsOfFile:(NSString *)path;
- (id)initWithContentsOfURL:(NSURL *)url;

- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile;
- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)atomically;
```

… и еще пара полезных методов

```
- (id)firstObject;
- (id)lastObject;
```

Перебор элементов коллекции

- В обычном цикле (подходит для массивов)
- С помощью цикла for-in (fast enumeration)
- С помощью блока

Перебор элементов коллекции при помощи цикла

```
NSArray *cities = @[@"Moscow", @"St. Petersburg", @"Kiev"];
for (int i = 0; i < [cities count]; i++) {
    NSLog(@"%@", cities[i]);
}
```

Перебор элементов коллекции при помощи fast enumeration

```
NSArray *cities = @[@"Moscow", @"St. Petersburg", @"Kiev"];
for (NSString *city in cities) {
    NSLog(@"%@", city);
}
```

ограничение: не поддерживает индексацию

Перебор элементов коллекции при помощи блока

```
NSArray *cities = @[@"Moscow", @"St. Petersburg", @"Kiev"];
[cities enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
        NSLog(@"%@", obj);
    }];
```

Fast enumeration & blocks

- Позволяют перебрать элементы *любых* коллекций.
- Fast enumeration поддерживает **break** и **continue**.
- Перебор элементов при помощи блока может быть прерван по средствам **BOOL *stop**.

BOOL *stop

Для того, чтобы прервать операцию перебора необходимо сделать следующее:

```
*stop = YES;
```

NSMutableArray

```
- (void)addObject:(id)anObject;
- (void)insertObject:(id)anObject atIndex:(NSUInteger)index;
- (void)removeLastObject;
- (void)removeObjectAtIndex:(NSUInteger)index;
- (void)replaceObjectAtIndex:(NSUInteger)index withObject:(id)anObject;
```

Mutable ⇔ Immutable

```
NSMutableArray *array = @[@1, @2, @3];
```

Можно ли так делать?

Справа стоит выражение типа **NSArray**. Так присваивать **нельзя** и компилятор об этом предупредит.

Mutable ⇔ Immutable

```
NSMutableArray *array = (NSMutableArray *)(@[@1, @2, @3]);
```

```
[array addObject:@4];
```

Что произойдёт теперь?

Предупреждение исчезло, а проблема нет: так все равно **нельзя** делать.

***** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '-[__NSArrayI addObject:]: unrecognized selector sent to instance 0x8d81480'**

Правильная реализация

```
NSMutableArray *array = [@[@1, @2, @3] mutableCopy];
[array addObject:@4];
```

или

```
NSMutableArray *array = [NSMutableArray arrayWithArray:@[@1, @2, @3]];
[array addObject:@4];
```

Задание для самостоятельного изучения

1. Прочитать документацию к NSArray,
2. Прочитать документацию к NSMutableArray.

NSDictionary + NSMutableDictionary

NSDictionary это...

- Структура данных, хранящая пары *ключ:значение*
- Ключи должны быть различны и должны удовлетворять протоколу **NSCopying**
- Обычно в качестве ключей берут **NSString**

Пример

```
NSDictionary<NSString *, NSNumber *> *weather = @{
        @"Moscow":@-3,
        @"Novosibirsk":@-12,
        @"Saint-Petersburg":@-7
    };
for (NSString *city in weather) {
    NSLog(@"%@ degrees in %@", weather[city], city);
}
```

Все ключи, все значения

```
[weather allKeys];
[weather allValues];
```

обратите внимание, что их порядок не обязан совпадать

NSMutableDictionary

```
- (void)removeObjectForKey:(id)aKey;
- (void)setObject:(id)anObject forKey:(id <NSCopying>)aKey;
```

Пример:

```
[dict setObject:object forKey:key]
```

или то же самое:

```
dict[key] = object
```

Добавить или изменить пару

```objc
NSMutableDictionary *weather = [[NSMutableDictionary alloc] init];
weather[@"Moscow"] = @-3;
//...
weather[@"Moscow"] = @-1;
```

Если по заданному ключу в словаре не было объекта, то ключ добавится, а если такой ключ уже был, то просто изменится значение по этому ключу.

Дополнительно

NSDictionary на Rypress

NSDictionary в документации Apple

NSSet + NSMutableSet

Для чего?

- Хранит неупорядоченный набор различных элементов.
- NSSet - неизменяемые объекты
- NSMutableSet - изменяемые объекты, можно производить с ними теоретико-множественные операции

Что с ним можно делать:

- Получить из NSArray и вернуть NSArray со всеми элементами (нельзя рассчитывать на определенный порядок)
- Проверять утверждения $x \in A$, $A \subseteq B$ и $A \cap B = \varnothing$
- Отфильтровать

Получить/вернуть

```
- (NSArray *)allObjects;
- (id)anyObject;

- (id)initWithArray:(NSArray *)array;
```

Теория множеств

$$x \in A$$

```
- (BOOL)containsObject:(id)anObject;
```

$$A \cap B = \varnothing$$

```
- (BOOL)intersectsSet:(NSSet *)otherSet;
```

$$A \subseteq B$$

```
- (BOOL)isSubsetOfSet:(NSSet *)otherSet;
```

Отфильтровать

```
- (NSSet *)objectsPassingTest:(BOOL (^)(id obj, BOOL *stop))predicate NS_AVAILABLE(10_6
- (NSSet *)objectsWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id obj,
```

Пример

```
NSSet *numbers = [NSSet setWithArray: @[@2, @3, @2, @10, @100, @50, @100]];
NSLog(@"%lu", (unsigned long)[numbers count]); // 5
NSSet *numbers2 = [NSSet setWithArray: @[@2, @3]];
NSLog(@"%d", [numbers2 isSubsetOfSet: numbers]); // 1
NSLog(@"%d", [numbers containsObject: @2]); // 1
NSLog(@"%d", [numbers2 intersectsSet: numbers]); // 1

NSSet *lessThan50 = [numbers objectsPassingTest:^BOOL(id obj, BOOL *stop) {
        return [obj intValue] < 50;
    }];
NSLog(@"%@", lessThan50); // 3, 10, 2 - порядок произвольный
```

NSMutableSet

```
- (void)addObject:(id)object;
- (void)removeObject:(id)object;

- (void)addObjectsFromArray:(NSArray *)array;
- (void)intersectSet:(NSSet *)otherSet;
- (void)minusSet:(NSSet *)otherSet;
- (void)removeAllObjects;
- (void)unionSet:(NSSet *)otherSet;

- (void)setSet:(NSSet *)otherSet;
```

Пример

```
NSMutableSet *cars = [NSMutableSet setWithArray: @[@"Porsche", @"Bugatti",
    @"Ferrari", @"Lada Kalina"]];
NSSet *otherCars = [NSSet setWithArray: @[@"Lamborghini", @"Ford"]];
NSSet *russianCars = [NSSet setWithObject: @"Lada Kalina"];
[cars unionSet: otherCars];
[cars minusSet: russianCars];

NSLog(@"%@", cars);
/*
Ford,
Porsche,
Ferrari,
Lamborghini,
Bugatti */
```

Дополнительно

NSSet в документации Apple

NSSet в Rypress

NSValue

Предназначение

An NSValue object is a simple container for a single C or Objective-C data item. It can hold any of the scalar types such as int, float, and char, as well as pointers, structures, and object ids. The purpose of this class is to allow items of such data types to be added to collections such as instances of NSArray and NSSet, which require their elements to be objects. NSValue objects are always immutable.

Может хранить что угодно (почти - см. следующий слайд), но в основном используется для хранения struct - для остального есть NSData и NSNumber.

Хранить можно не все

The type you specify must be of constant length. You cannot store C strings, variable-length arrays and structures, and other data types of indeterminate length in an NSValue—you should use NSString or NSData objects for these types. You can store a pointer to variable-length item in an NSValue object.

Получить/вернуть

```
– initWithBytes:objCType:
+ valueWithBytes:objCType:

+ valueWithNonretainedObject:
+ valueWithPointer:
+ valueWithPoint:
+ valueWithRange:
+ valueWithRect:
+ valueWithSize:

– getValue:
– objCType

– nonretainedObjectValue
– pointerValue
– pointValue
– rangeValue
– rectValue
– sizeValue
```

valueWithBytes:objCType:

```
+ (NSValue *)valueWithBytes:(const void *)value objCType:(const char *)type;
```

где взять этот самый objCType?

@encode

В Objective-C у любого типа существует его внутреннее представление в виде строки C.

Его можно получить с помощью **@encode**

Например, **@encode(id) == "@"**

getValue:

```
- (void)getValue:(void *)value;
```

- копирует значение в заданный буфер. Он должен быть достаточно большого размера.

Пример

```
typedef struct Str {
    int x;
    char y;
} MyStruct;
MyStruct s = (MyStruct){.x = 100, .y = 'x'};
NSValue *val = [NSValue value:&s withObjCType:@encode(MyStruct)];

MyStruct s2;
[val getValue:&s2];

NSLog(@"%i, %c", s2.x, s2.y);
```

Полезные методы

```
+ valueWithPoint: // для CGPoint
+ valueWithRange: // для NSRange
+ valueWithRect: // для CGRect
+ valueWithSize: // для CGSize

– pointValue
– pointerValue
– rangeValue
– rectValue
```

Дополнительно

http://nshipster.com/nsvalue/

http://nshipster.com/type-encodings/

NSValue в документации Apple

NSNumber

Для чего используется

- В основном, чтобы передать в метод число как объект (или получить обратно).
- В частности, для того, чтобы сохранить число в **NSArray**, **NSDictionary** или **NSSet**.
- Объекты класса **NSNumber** неизменяемые.

Раньше их создавали так

```
NSNumber *pi = [NSNumber numberWithFloat:3.1415];
NSNumber *e = [NSNumber numberWithFloat:2.71];
```

долго и нудно (плюс лишняя пара квадратных скобок)

Теперь создают так

```
NSNumber *pi = @3.1415;
NSNumber *e = @2.71;

NSNumber *ePlusPi = @(3.1415 + 2.71);
```

на самом деле, тут продолжает вызываться метод *numberWithFloat:* или подобный, просто теперь компилятор это делает сам

Получить число обратно

```
- (char)charValue;
- (unsigned char)unsignedCharValue;
- (short)shortValue;
- (unsigned short)unsignedShortValue;
- (int)intValue;
- (unsigned int)unsignedIntValue;
- (long)longValue;
- (unsigned long)unsignedLongValue;
- (long long)longLongValue;
- (unsigned long long)unsignedLongLongValue;
- (float)floatValue;
- (double)doubleValue;
- (BOOL)boolValue;
- (NSInteger)integerValue NS_AVAILABLE(10_5, 2_0);
- (NSUInteger)unsignedIntegerValue NS_AVAILABLE(10_5, 2_0);
- (NSString *)stringValue;
```

Что еще можно делать?

Также NSNumber можно сравнивать

```
- (NSComparisonResult)compare:(NSNumber *)otherNumber;
- (BOOL)isEqualToNumber:(NSNumber *)number;
```

Ну и, собственно, все.

Вопрос

```
NSNumber *pi = @3.1415;
NSLog(@"%?", pi);
```

Какой format specifier нужно подставить вместо **%?** ?

Не правильно, нужно использовать **%@**, так как NSNumber — это объект.

Задания

1. Узнать, что такое **NS_AVAILABLE**
2. Что будет, если поместить в NSNumber число одного типа, а обратно пытаться получить другое?

```
NSNumber *e = @2.71;
[e intValue];
[e boolValue];
```

3. Почитать про NSDecimalNumber

NSDecimalNumber на Rypress

NSData + NSMutableData

Что это?

Класс, позволяющий хранить бинарные данные. Многие методы для работы с интернетом возвращают **NSData** в качестве результата.

bytes & length

В целом **NSData** — это указатель на данные *bytes* и размер данных *length* (в байтах).

```
- (id)initWithBytes:(const void *)bytes
    length:(NSUInteger)length;

- (NSUInteger)length;
- (const void *)bytes NS_RETURNS_INNER_POINTER;
```

Пример

```
int x = 2014;
NSData *data = [NSData dataWithBytes:&x length:sizeof(x)];
NSLog(@"%@", data); // de070000
```

Инициализация NSData

```
- (id)initWithBytes:(const void *)bytes length:(NSUInteger)length;
- (id)initWithBytesNoCopy:(void *)bytes length:(NSUInteger)length;
- (id)initWithBytesNoCopy:(void *)bytes
    length:(NSUInteger)length
    freeWhenDone:(BOOL)b;
- (id)initWithContentsOfFile:(NSString *)path
    options:(NSDataReadingOptions)readOptionsMask
    error:(NSError **)errorPtr;
- (id)initWithContentsOfURL:(NSURL *)url
    options:(NSDataReadingOptions)readOptionsMask
    error:(NSError **)errorPtr;
- (id)initWithContentsOfFile:(NSString *)path;
- (id)initWithContentsOfURL:(NSURL *)url;
- (id)initWithData:(NSData *)data;
```

Сохранение в файл/URL

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile;
- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)atomically;
- (BOOL)writeToFile:(NSString *)path
    options:(NSDataWritingOptions)writeOptionsMask
    error:(NSError **)errorPtr;
- (BOOL)writeToURL:(NSURL *)url
    options:(NSDataWritingOptions)writeOptionsMask
    error:(NSError **)errorPtr;
```

Поиск/получение подданных

```
- (NSRange)rangeOfData:(NSData *)dataToFind
    options:(NSDataSearchOptions)mask
    range:(NSRange)searchRange;

- (NSData *)subdataWithRange:(NSRange)range;
```

NSDataSearchOptions

```
typedef NS_OPTIONS(NSUInteger, NSDataSearchOptions) {
    NSDataSearchBackwards = 1UL << 0,
    NSDataSearchAnchored = 1UL << 1
};
```

NSDate

NSDate — это...

- Специальный класс для хранение абсолютного значения времени, не зависящего от конкретной системы исчисления или от временной зоны.
- Неизменяемый объект.

Операции, доступные для NSDate

- Сравнение дат
- Преобразование даты в строку
- Увеличение/уменьшение даты

Создание NSDate

```
NSDate *currentDate = [NSDate date]; // Создание текущей даты

NSTimeInterval dayInterval = 24 * 60 * 60; // Временной интервал, равный одним суткам
NSDate *tomorrowDate = [[NSDate alloc] initWithTimeIntervalSinceNow:dayInterval];
// Создание завтрашней даты
```

Сравнение дат

```
BOOL equalDates = [currentDate isEqualToDate:anotherDate];
NSDate *earlierDate = [currentDate earlierDate:anotherDate];
NSDate *laterDate = [currentDate laterDate:anotherDate];
```

Преобразование даты в строку

```
NSDateFormatter *dateFormatter = [NSDateFormatter new];
[dateFormatter setDateStyle:kCFDateFormatterMediumStyle];
[dateFormatter setTimeStyle:kCFDateFormatterShortStyle];
NSString *dateAsString = [dateFormatter stringFromDate:currentDate];
```

Увеличение/уменьшение даты

```
NSTimeInterval dayInterval = 24 * 60 * 60;
NSDate *currentDate = [NSDate date];
NSDate *tomorrowDate = [currentDate dateByAddingTimeInterval:dayInterval];
```

Задание для самостоятельного изучения

- Изучить покомпонентное задание даты при помощи классов **NSCalendar** и **NSDateComponents**. Для этого воспользоваться статьёй.

NSError + NSException

Ошибки бывают двух видов

1. Деление на 0, выход за границы массива, ...
   NSException
2. Не удалось загрузить файл, не удалось создать объект, …
   NSError

NSError

```
NSURL *yandex = [NSURL URLWithString:@"http://ya.ru"];
NSError *error = nil;
NSString *yandexString = [NSString stringWithContentsOfURL:yandex
    encoding:NSUTF8StringEncoding error:&error];
```

NSError всегда передается как указатель на указатель

Информация об ошибке

```
- (NSInteger)code; // код ошибки
- (NSString *)domain;  // домен ошибки (напр., NSCocoaErrorDomain)
// могут существовать ошибки с одним кодом, но разными доменами

- (NSDictionary *)userInfo;  // дополнительная информация
```

userInfo: полезные ключи

- NSLocalizedDescriptionKey
- NSLocalizedFailureReasonErrorKey
- NSLocalizedRecoverySuggestionErrorKey

Как использовать такие методы?

```
//...
NSError *error = nil;
NSString *yandexString = [NSString stringWithContentsOfURL:yandex
    encoding:NSUTF8StringEncoding error:&error];

// сначала проверяем, произошла ли ошибка...
if (error != nil) {
    // …, и только после этого обрабатываем ошибку
    NSLog(@"Error - %@", error);
}
```

Как самому создать такой метод?

```
- (BOOL)myOwnMethodReturnsError:(NSError *__autoreleasing *)error {
    //...
    if (somethingWrong && error) {
        *error = [[NSError alloc] initWithDomain:@"MyErrorDomain"
            code:666 userInfo:nil];
        return NO;
    }
    return YES;
}
```

NSException

```
NSArray *array = @[@"one", @"two", @"three"];
int index = 100;
@try {
    NSLog(@"%d item: %@", index, array[index]);
}
@catch (NSException *exception) {
    NSLog(@"Oops... exception occured");
    NSLog(@"Name - %@", exception.name);
    NSLog(@"Reason - %@", exception.reason);
}
@finally {
    // запускается независимо от того, было исключение или нет
}
```

Скорее всего, вам это не пригодится

```
NSException *myException = [[NSException alloc] initWithName:@"ExceptionName"
    reason:@"ExceptionReason" userInfo:nil];
@throw myException;

// или
[NSException raise:@"ExceptionName" format:@"ExceptionReason"];
```

Дополнительные материалы

http://nshipster.com/nserror/

http://rypress.com/tutorials/objective-c/exceptions.html

Бонусная часть

О чем будем говорить

- Object Subscripting
- NSCopying, isEqual:, hash

Object Subscripting

```
NSMutableArray *array = [@[@1,@2,@3] mutableCopy];
NSLog(@"%@", array[1]);
array[2] = @"Bla-bla-bla";
```

```
NSMutableDictionary *dictionary = [@{@1: @"one", @2: @"two"} mutableCopy];
NSLog(@"%@", dictionary[@1]);
dictionary[@3] = @"three";
```

Обращение к элементам коллекций идёт через квадратные скобки.

Хотите так же?

Это может быть полезно, если ваш класс хранит коллекцию элементов (например, шахматная доска, судоку, граф, …)

```
// Sudoku *sudoku = ...
NSNumber *number = sudoku[@"1,3"];

// Graph *graph = ...
Vertex *vertex = graph[4];
```

Все сводится к вызову методов

```
NSMutableArray *array = [@[@1,@2,@3] mutableCopy];
NSLog(@"%@", [array objectAtIndexedSubscript:1]);
[array setObject:@"Bla-bla-bla" atIndexedSubscript:2];

NSMutableDictionary *dictionary = [@{@1: @"one", @2: @"two"} mutableCopy];
NSLog(@"%@", [dictionary objectForKeyedSubscript:@2]);
[dictionary setObject:@"three" forKeyedSubscript:@3];
```

Ага, вот эти методы

```
// myObject[idx];
- (id)objectAtIndexedSubscript:(NSUInteger)idx;

// myObject[idx] = obj;
- (void)setObject:(id)obj atIndexedSubscript:(NSUInteger)idx;

// myObject[key];
- (id)objectForKeyedSubscript:(id <NSCopying>)key;

// myObject[key] = obj;
- (void)setObject:(id)obj forKeyedSubscript:(id <NSCopying>)key;
```

чтобы использовать удобный синтаксис, реализуйте один или несколько методов из приведенных выше

NSCopying — что это?

Мы его уже встречали 2 раза:

- На прошлом слайде
- У **NSDictionary** ключ должен удовлетворять **NSCopying**

Из документации

The ***NSCopying*** protocol declares a method for providing functional copies of an object. The exact meaning of “copy” can vary from class to class, but a copy must be a functionally independent object with values identical to the original at the time the copy was made.

NSString и **NSNumber** поддерживают этот протокол. А как реализовать его в своем классе?

Единственный метод

```
@protocol NSCopying

- (id)copyWithZone:(NSZone *)zone;

@end
```

Как реализовать?

- Если суперкласс поддерживает **NSCopying**, вызывайте *[super copyWithZone:zone]*;
- Если нет, используйте *alloc-init*
- Если объект в принципе неизменяемый, можно вернуть *self*

Пример

```
@interface Person : NSObject <NSCopying>
@property (copy, nonatomic) NSString *name;
@property (copy, nonatomic) NSString *surname;
@property (copy, nonatomic) NSUInteger age;
@end


@implementation Person

- (id)copyWithZone:(NSZone *)zone {
    Person *personCopy = [[Person alloc] init];
    personCopy.name = self.name;
    personCopy.surname = self.surname;
    personCopy.age = self.age;
    return personCopy;
}

@end
```

И ради чего это?

Теперь можно использовать класс **Person** как ключ в **NSDictionary**.

Иногда это бывает нужно.

Равенство объектов

Есть много ситуаций, когда требуется проверить, равны объекты или нет. Например:

- когда добавляем пару ключ:значение в dictionary
- когда добавляем элемент в множество
- когда ищем объект (*indexOfObject:* у *NSArray*)

NSObject

Для этих проверок у **NSObject** есть 2 метода:

```
- (NSUInteger)hash;
- (BOOL)isEqual:(id)object;
```

Hash

- Хэш — это число.
- Если хэши различны, объекты точно не совпадают.
- Если одинаковы — неизвестно. Тогда можно проверить с помощью метода *-isEqual:*.

isEqual:

в **NSObject** *isEqual:* реализован так:

```
- (BOOL)isEqual:(id)other
{
    return self == other;
}
```

Т.е. объект эквивалентен только сам себе.

Если это не то, что мы хотим, нужно самому переопределить этот метод.

Как реализовать их самому?

На эту тему есть хорошая статья.

Если коротко

- *isEqual:* — обычно достаточно проверить на равенство все свойства

```
- (BOOL)isEqual:(id)other
{
    if (![other isKindOfClass:[self class]]) {
        return NO;
    }

    return ([[other name] isEqualToString:self.name] &&
        [[other surname] isEqualToString:self.surname] &&
        [other age] == self.age);
}
```

- *hash* — можно взять хэши от всех свойств и поксорить их.

```
- (NSUInteger)hash {
    return [self.name hash] ^ [self.surname hash] ^ self.age;
}
```

(Однако это может быть не самый лучший способ — см. статью)

И самое главное

Если вы переопределили *-isEqual:*, **обязательно** нужно переопределить *hash*.

Дополнительно

Object Subscripting

Implementing Equality and Hashing