



Лекция 1.0

Swift как язык программирования

Swift - основы

Кто Я?

Руководитель службы разработки мобильных приложений. Создаю iOS приложения с 2009 года.

В недавнем прошлом участвовал в разработке масштабных систем для Федеральной таможенной службы и Банка России.

В Яндексе с апреля 2016 года.



Что будем изучать?

- › Swift
- › Cocoa Touch
- › Напишем свое приложение "Заметки"

Swift



Swift - Уже везде



iOS



macOS



Servers (Linux)

Swift, как насчет Legacy?



Swift



OBJECTIVE-C

Objective-C

Что нам потребуется



Xcode



Mac



Developer Program

Начнем



Со Swift конечно :)

SwiftBook



English



Русская версия

Инстанцируем всякие штуки

```
let iAmConstant = 10  
iAmConstant = 20 // Forbidden
```

```
var iAmVariable = 10  
iAmVariable = 20 // OK
```

```
var view = UIView()  
var view2: UIView? = UIView()
```

Поля и методы

```
let car = Car()  
  
print(car.model) // using field  
print(car.mark) // using another field  
print(car.dump()) // using function
```


Ветвление

```
if car.mark == "Honda" {  
    print("Hurray, I love Honda")  
} else if car.model == "Integra" {  
    print("Strange, Integra, but not Honda?")  
} else {  
    print("Oh no, Where is my Honda")  
}
```

Циклы

```
for (i = 0, i < 100; i++) { ... }
```

```
for i in 0..<100 { }
```

```
for i in 0...100 { }
```

```
var i = 0  
while i < 100 {  
    i += 1  
}
```

```
var j = 0  
repeat {  
    j += 1  
} while j < 100
```

Циклы foreach

```
let array = [1, 2, 3, 4]
```

```
for number in array {  
}
```

```
for (index, number) in array.enumerated() {  
}
```

```
let dictionary = [1: "One", 2: "Two", 3: "Three"]
```

```
for (number, representation) in dictionary {  
}
```


Функции

```
func doSomething() {  
}
```

```
func doSomething() -> Int {  
    return 1  
}
```

Функции

```
func doSomething(_ good: String) {  
    print(good)  
}  
doSomething("Present a present")
```

```
func doSomething(bad: String) {  
    print(bad)  
}  
doSomething(bad: "Steal a thing")
```

```
func doSomething(ugly bad: String) {  
    print(bad)  
}  
doSomething(ugly: "Go away")
```

Функции

```
func doTheSum(arg1: Int, arg2: Int) -> Int {  
    return arg1 + arg2  
}  
print(doTheSum(arg1: 1, arg2: 2))
```

```
func doTheSubtract(arg1: Int = 0, arg2: Int = 0) -> Int {  
    return arg1 - arg2  
}  
print(doTheSubtract())  
print(doTheSubtract(arg1: 1))  
print(doTheSubtract(arg2: 2))  
print(doTheSubtract(arg1: 2, arg2: 1))
```


Ветвление, guard

```
func checkTwo(a: Int) -> Bool {  
    guard a == 2 else {  
        return false  
    }  
  
    return true  
}
```

```
print(checkTwo(a: 1)) // false  
print(checkTwo(a: 2)) // true
```

Closures

```
var lambda1 = {  
    print("Hi")  
}  
lambda1()
```

```
var lambda2 = { (_ name: String) in  
    print("Hi, \(name)")  
}  
lambda2("Denis")
```

```
var lambda3 = { () -> String in  
    return UUID().uuidString  
}  
print(lambda3())
```

Класс

```
class Car {  
    let uid: String = UUID().uuidString  
    let mark: String  
    let model: String  
  
    var mileage: Int = 0  
  
    init(mark: String, model: String, mileage: Int = 0) {  
        self.mark = mark  
        self.model = model  
  
        self.mileage = mileage  
    }  
  
    func go(for kilometers: Int) {  
        mileage += kilometers  
    }  
}
```


Класс, модификаторы доступа

```
class Car {  
    public var mileage: Int = 0  
    private var owner: String = "Pechkin"  
    private(set) var guardian: String = "Sentinel"  
    fileprivate var uid: String = UUID().uuidString  
}
```

```
extension Car {  
    func dump() -> String {  
        var parts = [String]()  
        parts.append("mileage: \(mileage)")  
parts.append("owner: \(owner)")  
        parts.append("guardian: \(guardian)")  
        parts.append("uid: \(uid)")  
        return parts.join(separator: "\n")  
    }  
}
```

Класс, инициализаторы и деинициализатор

```
class Vehicle { }  
class Car : Vehicle {  
    let mark: String  
    let model: String  
  
    convenience override init() {  
        self.init(mark: "Honda", model: "Integra")  
    }  
  
    init(mark: String, model: String) {  
        self.mark = mark  
        self.model = model  
  
        super.init()  
    }  
  
    deinit {  
        print("Car will be disposed")  
    }  
}
```

Интерфейс

Кто знает, что такое?

Протокол

```
protocol Dumpable {  
    var isDumpable: Bool { get }  
  
    func dump() -> String  
}  
  
class Car: Dumpable {  
    var isDumpable: Bool = true  
  
    func dump() -> String {  
        return "Car dump"  
    }  
}
```

Расширения (в objс - категории)

```
extension String {  
    func calculate(letter: String) -> Int {  
        var sum = 0  
        for char in characters {  
            if "\(char)" == letter {  
                sum += 1  
            }  
        }  
        return sum  
    }  
}
```

```
print("Baadaaboom".calculate(letter: "a"))
```


Optionals

```
var aThing: String = ""
```

```
var maybeThing: String? = nil  
maybeThing = "A thing"
```

Optionals

```
let view1 = UIView()  
let view2: UIView? = UIView()  
  
print(view1.frame) // (0.0, 0.0, 0.0, 0.0)  
  
print(view2?.frame) // Optional((0.0, 0.0, 0.0, 0.0))  
print(view2!.frame) // Optional((0.0, 0.0, 0.0, 0.0))
```

Optionals

```
let view1: UIView? = nil
let view2: UIView? = UIView()

print(view1?.frame) // nil
print(view2?.frame) // Optional((0.0, 0.0, 0.0, 0.0))

print(view1!.frame) // CRASH!
print(view2!.frame) // Optional((0.0, 0.0, 0.0, 0.0))
```

| Крайне опасно
использовать Force
Unwrapping Optional

То есть opt!

Optionals, if let

```
let view1: UIView? = nil
let view2: UIView? = UIView()

if let view = view1 { // No enter, view1 == nil
    print(view.frame)
}
if let view = view2 { // ok, view2 is ok
    print(view.frame) // prints (0.0, 0.0, 0.0, 0.0)
}
```


Optionals, guard

```
func printFrame(view: UIView?) {  
    guard let view = view else {  
        print("No view")  
        return  
    }  
  
    print(view.frame)  
}  
  
let view1: UIView? = nil  
let view2: UIView? = UIView()  
  
printFrame(view: view1) // No view  
printFrame(view: view2) // (0.0, 0.0, 0.0, 0.0)
```

Golden Path

Используйте guard умело



ARC

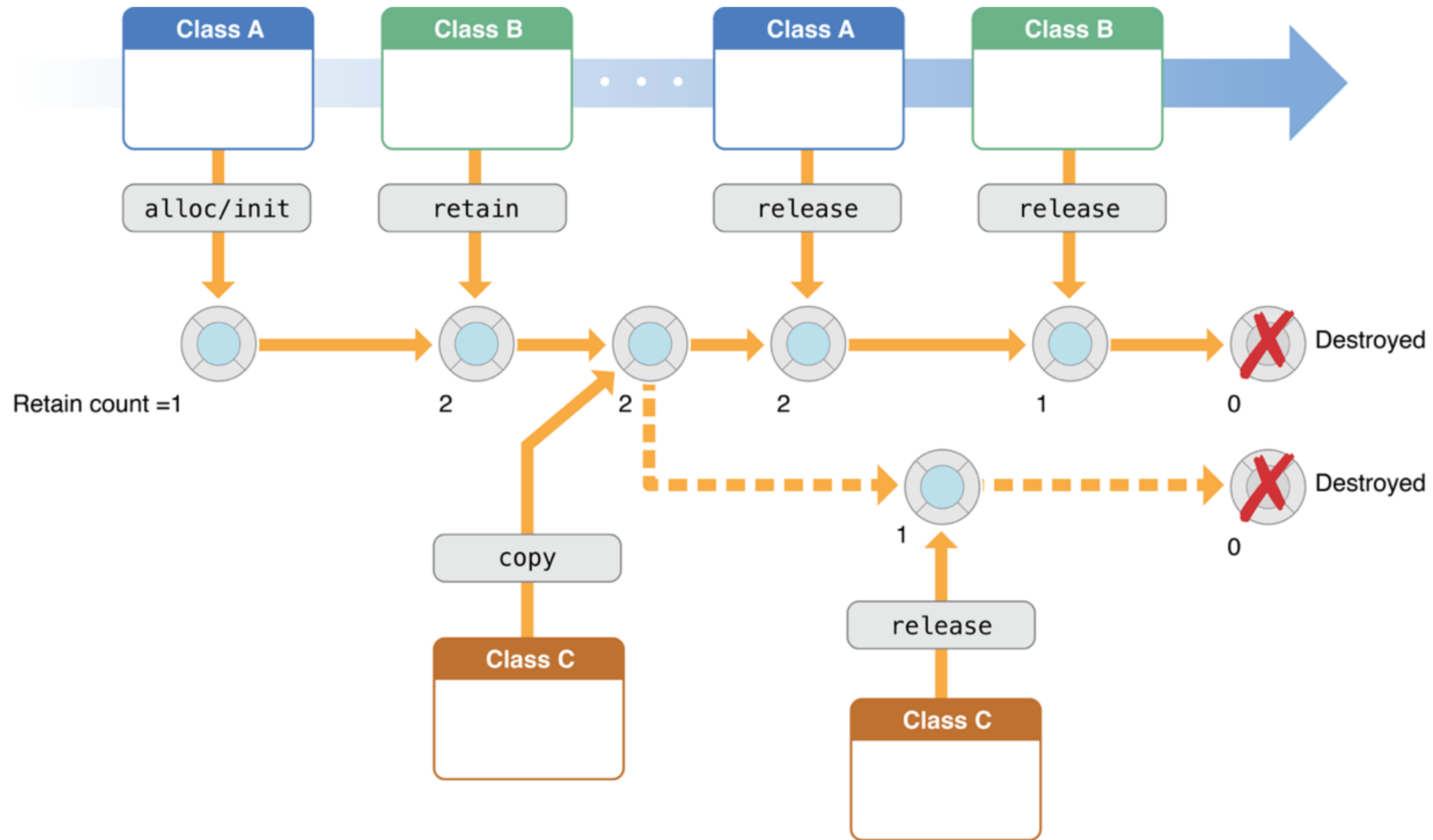
Кто знает, что такое? MRC?


MRC

- › RefCount
- › alloc (+1)
- › retain, copy (+1)
- › release (-1)
- › autorelease (-1, deferred)



Reference Counting

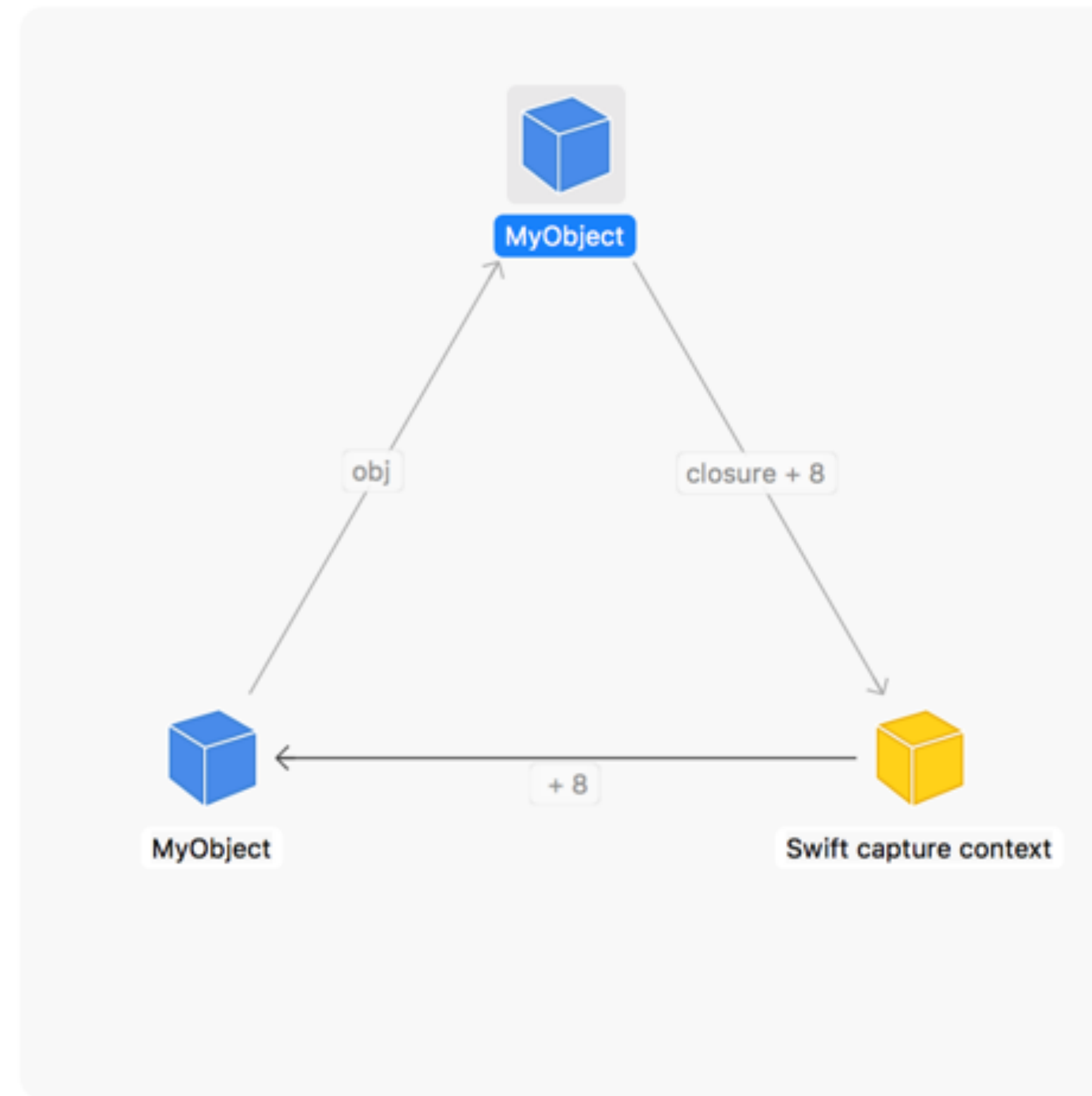




ARC делает управление
памятью за вас

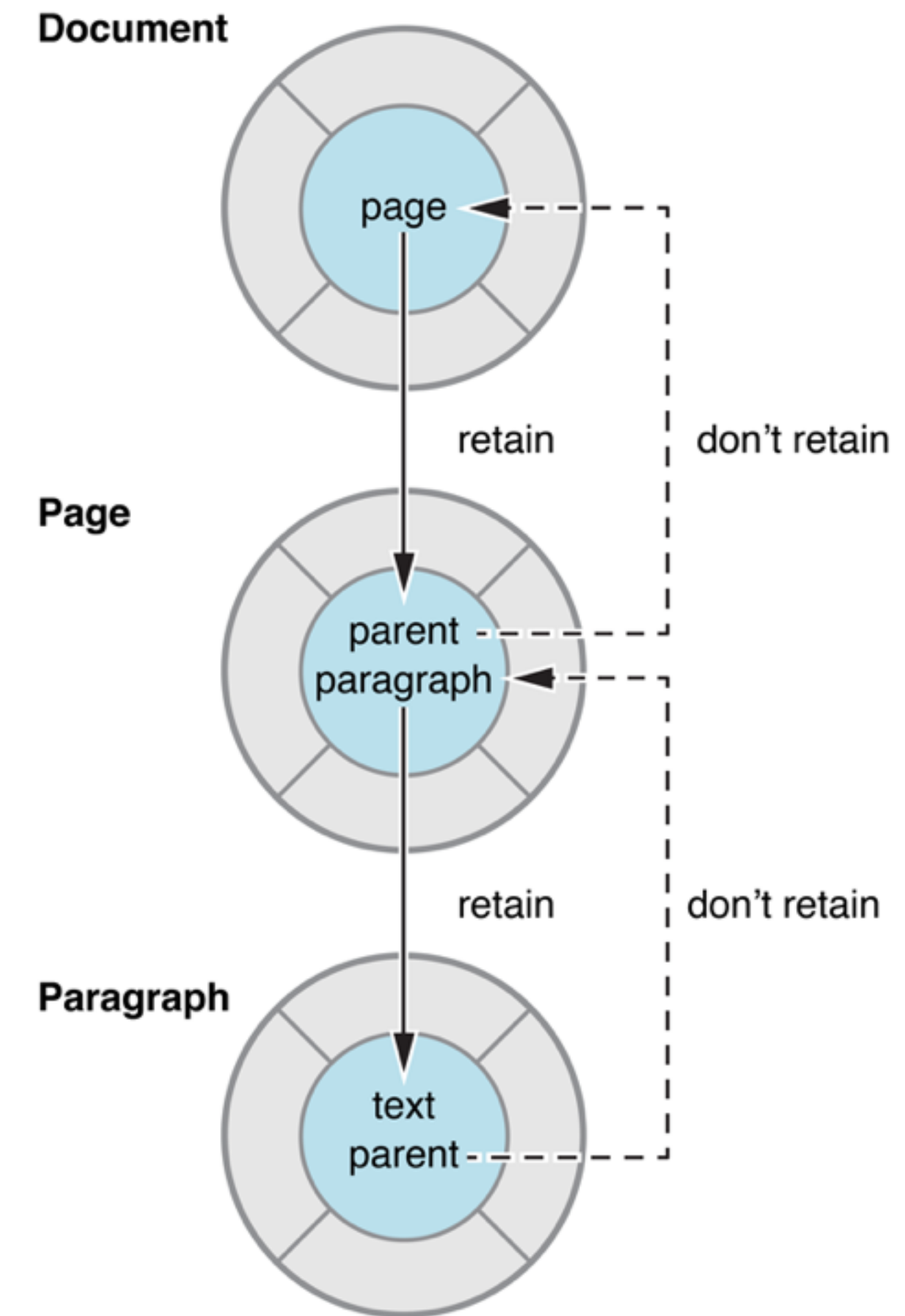
Но будьте осторожны (!!!), это не GC


ARC, retain cycle



ARC, разбираем retain cycle

- › Чтобы владеть объектом - strong
- › Чтобы ссылаться на объект - weak





Reference Type vs Value Type

class vs struct

Struct

```
struct Car {  
    let mark: String  
    let model: String  
}
```

```
print(Car(mark: "Honda", model: "Integra"))
```

Struct

- › Нет наследования
- › Нет type casting
- › Нет deinit (деинициализатора)
- › Нет RefCount
- › **Всегда копируются**

Struct, стандартная библиотека

› Double, Int, Float, ...

› String (!!!)

› Array []

› Dictionary [:]

Array

```
let array1 = [1, 2, 3, 4] // [Int] with [1, 2, 3, 4]
let array4 = [Int]() // [Int] with []

let array2: [Int] = [1, 2, 3, 4] // [Int] with [1, 2, 3, 4]
let array3: [Int] = [] // [Int] with []

print(array1[0]) // 1
print(array1[2]) // 3
```

Array, mutable

```
var array = [Int]()  
print(array) // []
```

```
array.append(1)  
print(array) // [1]
```

```
array.insert(2, at: 0)  
print(array) // [2, 1]
```

```
array.remove(at: 0)  
print(array) // [1]
```

```
array.removeAll()  
print(array) // []
```


Dictionary

```
// [Int: String] with [1: "One", 2: "Two"]  
let dictionary1 = [1: "One", 2: "Two"]  
  
// [Int: String] with [:]  
let dictionary2 = [Int: String]()  
  
// [Int: String] with [1: "One", 2: "Two"]  
let dictionary3: [Int: String] = [1: "One", 2: "Two"]  
  
// [Int: String] with [:]  
let dictionary4: [Int: String] = [:]  
  
print(dictionary1[1]) // Optional("One") !!!!
```

Dictionary, mutable

```
var dict = [Int: String]()  
print(dict) // [:]  
  
dict[1] = "One"  
print(dict) // [1: "One"]  
print(dict[1]) // Optional("One")  
  
dict[1] = nil  
print(dict) // [:]
```

Enums

```
enum Planets {  
    case mercury  
    case venus  
    case earth  
    case mars  
    case jupiter  
    case saturn  
    case uranus  
    case neptune  
    // NOTE: no pluto :(  
}
```

Enums, typed

```
enum Planets: Int {  
    case mercury = 1  
    case venus = 2  
    case earth = 3  
    case mars = 4  
    case jupiter = 5  
    case saturn = 6  
    case uranus = 7  
    case neptune = 8  
    // NOTE: no pluto :(  
}  
  
print(Planets.saturn.rawValue) // 6
```

Enums are objects

```
enum Planets: Int {  
    var isInnerSystem: Bool {  
        switch self {  
            case .mercury, .venus, .earth, .mars: return true  
            default: return false  
        }  
    }  
  
    func distanceTo(other planet: Planets) -> Double {  
        return Double(arc4random()) // TODO: proper distance  
    }  
}  
  
print(Planets.mars.isInnerSystem) // true  
print(Planets.venus.distanceTo(other: .jupiter)) // some big number
```

Enums + values

```
enum ApiCallResult {  
    case success(data: Data)  
    case failure(statusCode: Int, reason: String)  
}  
  
// success(0 bytes)  
print(ApiCallResult.success(data: Data()))  
  
// failure(500, "Server error")  
print(ApiCallResult.failure(statusCode: 500, reason: "Server error"))
```

Pattern matching

Кто знает, что такое?

Simple Pattern Matching

```
let bool1 = true
let bool2 = false

switch (bool1, bool2) {
  case (false, false): print("0")
  case (false, true): print("1")
  case (true, false): print("2")
  case (true, true): print("3")
}
```


Pattern Matching, Wildcard

```
let p: String? = nil
switch p {
  case _?: print ("Has String")
  case nil: print ("No String")
}
```

```
switch (15, "example", 3.14) {
  case (_, _, let pi): print ("pi: \(pi)")
}
```

Pattern Matching, Value binding

```
switch (4, 5) {  
  case let (x, y): print("\(x) \(y)")  
}
```

Pattern Matching, Tuples

```
let age = 23
let job: String? = "Operator"
let payload: AnyObject = NSDictionary()

switch (age, job, payload) {
    case (let age, _?, _ as NSDictionary):
        print(age)
    default: break
}
```

Pattern Matching, Enum case

```
enum Entities {  
    case soldier(x: Int, y: Int)  
    case tank(x: Int, y: Int)  
    case player(x: Int, y: Int)  
}  
  
let entities: [Entities] = [.tank(x: 1, y: 1), .soldier(x: 10, y: 1)]  
  
for e in entities {  
    switch e {  
    case let .soldier(x, y):  
        print("soldier at [(x); (y)]")  
    case let .tank(x, y):  
        print("tank at [(x); (y)]")  
    case let .player(x, y):  
        print("player at [(x); (y)]")  
    }  
}
```

Pattern Matching, Casting

```
let a: Any = 5
switch a {
  case is Int: print ("Int")
  case let n as Int: print (n + 1)
  default: break
}
```

Pattern Matching, Expressions

```
switch 5 {  
  case 0...10: print("In range 0-10")  
  default: break  
}
```

Pattern Matching, Expressions

```
struct Soldier {  
    let hp: Int  
    let x: Int  
    let y: Int  
}  
  
func ~= (pattern: Int, value: Soldier) -> Bool {  
    return pattern == value.hp  
}  
  
let soldier = Soldier(hp: 0, x: 10, y: 10)  
switch soldier {  
    case 0: print("dead soldier")  
    default: ()  
}
```

Pattern matching like a Boss

```
func fibonacci(_ i: Int) -> Int {  
    switch(i) {  
        case let n where n <= 0: return 0  
        case 0, 1: return 1  
        case let n: return fibonacci(n - 1) + fibonacci(n - 2)  
    }  
}  
  
print(fibonacci(8))
```




Generics

Generic Struct

```
struct Pack<Obj> {  
    fileprivate var objects: [Obj] = []  
  
    mutating func pack(_ object: Obj) {  
        objects.append(object)  
    }  
  
    mutating func unpack() -> Obj? {  
        guard !objects.isEmpty else { return nil }  
        return objects.remove(at: 0)  
    }  
}
```

Generic Struct

```
var bag = Pack<String>()
```

```
bag.pack("Drill")
```

```
bag.pack("Bolts")
```

```
print(bag.unpack()) // Optional("Drill")
```

```
print(bag.unpack()) // Optional("Bolts")
```

```
print(bag.unpack()) // nil
```

```
print(bag.unpack()) // nil
```

Generic Struct, Constrain

```
protocol Packable {  
  
}  
  
struct Pack<Obj: Packable> {  
    ...  
}  
  
var bag = Pack<String>() // Error – String is not packable
```

Generic Struct, Constrain

```
protocol Packable {  
}  
  
extension String: Packable {  
}  
  
struct Pack<Obj: Packable> {  
    ...  
}  
  
var bag = Pack<String>() // Ok, now String is Packable
```

Вопросы ?



It's a DEMO time



Задача



Общие требования

- › Реализовать структуру Note (требования ниже)
- › Реализовать расширение Note для разбора json (требования ниже)
- › Реализовать класс FileNotebook (требования ниже)
- › Реализовать сохранение и загрузку FileNotebook в файл и из файла

Note

- › Иммутабельная структура
- › Содержит уникальный идентификатор `uid`, если не задан пользователем - генерируется (`UUID().uuidString`)
- › Содержит обязательные строковые поля - `title` и `content`
- › Содержит цвет заметки, пользователь структуры может его не задать, тогда белый по умолчанию (Класс `UIColor` из `UIKit`)
- › Содержит обязательное поле важность, должно быть `enum`, может иметь три варианта - "неважная", "обычная" и "важная"

Note, parsing json

- › Расширение для структуры Note
- › Содержит функцию (static func parse(json: [String: Any]) -> Note?) для разбора json
- › Содержит вычисляемое свойство (var json: [String: Any]) для формирования json'a
- › Цвет сохранять в json только, если он не белый
- › Не сохранять в json важность, если она "обычная"
- › Не сохранять в json сложные объекты (UIColor, Date)

FileNotebook

- › Содержал закрытую для внешнего изменения, но открытую для получения коллекцию Note
- › Содержал функции добавления новой заметки
- › Содержал функцию удаления заметки (на основе uid)
- › Содержал функцию сохранения всей записной книжки в файл
- › Содержал функцию загрузки записной книжки из файла

Усложнения

› Реализовать самоуничтожение заметок, в разумные сроки конечно

DummyNotebook, получить путь куда можно save

```
static var filePath: String? {  
    guard let dir =  
NSSearchPathForDirectoriesInDomains(.documentDirectory, .allDomainsMask,  
true).first else {  
        return nil  
    }  
  
    let path = "\(dir)/notes.plist"  
    print(path)  
  
    return path  
}
```

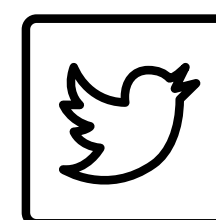
Контакты

Малых Денис

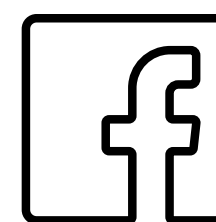
iOS Developer



mrdekk@yandex.ru



@mrdekk



mrdekk

Спасибо за внимание

