Movile Next

(iOS)

Apresentação


30
MARCELO

Marcelo Fabri

- @marcelofabri_
- Na Movile desde 2012
- Unicamp
- COTUCA

Objetivos

- Ensinar o básico de iOS
- Dar o pontapé inicial
- Mostrar um pouco como trabalhamos

Dinâmica

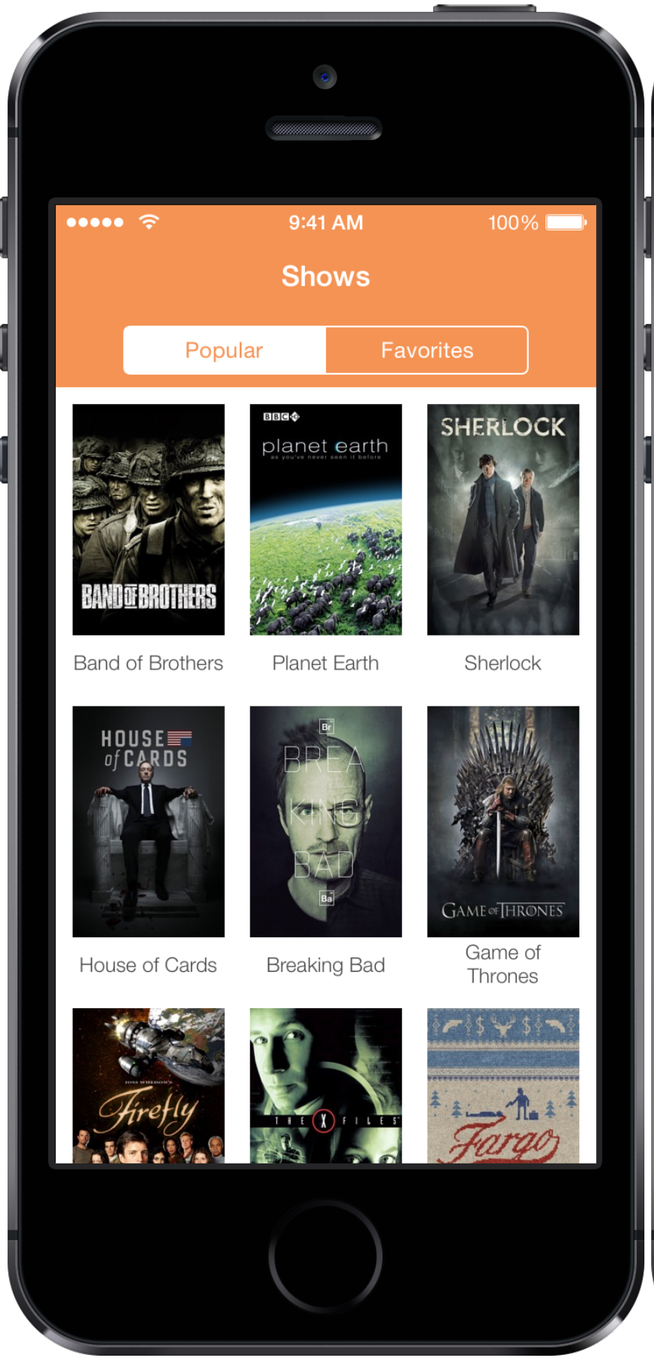
- Apresentações teóricas (não muito longas)
- Projeto prático durante o curso

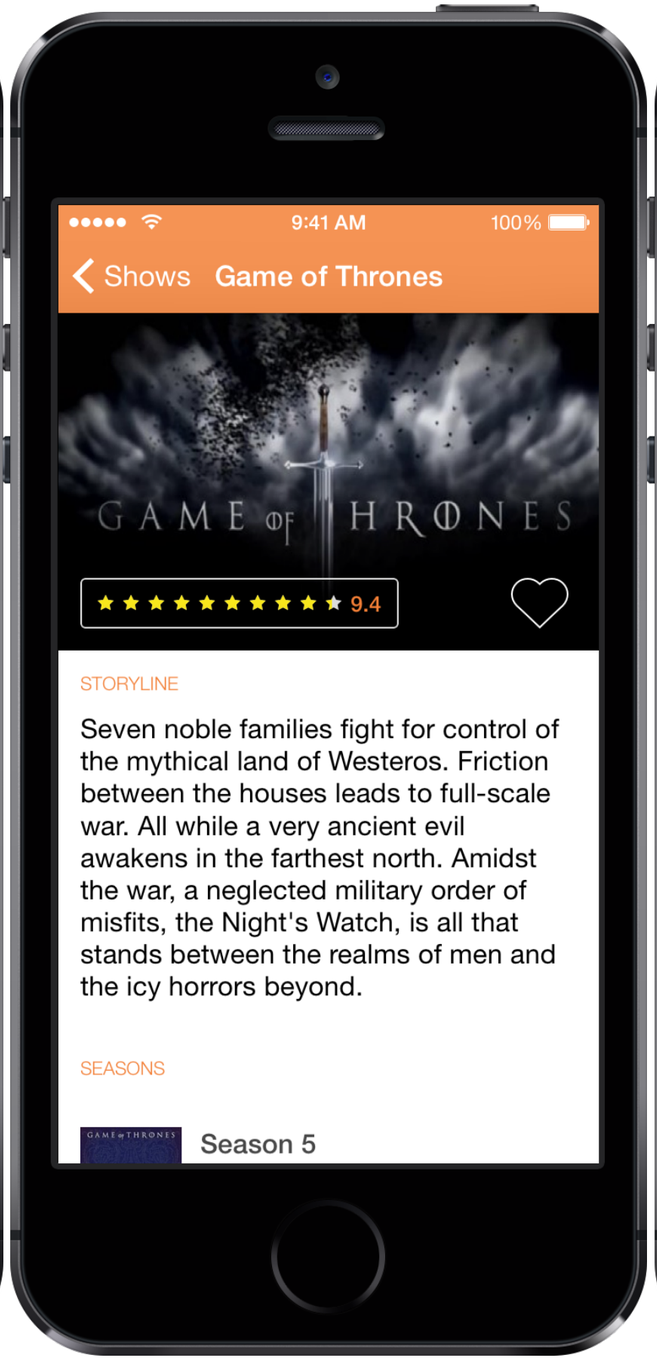
Contato

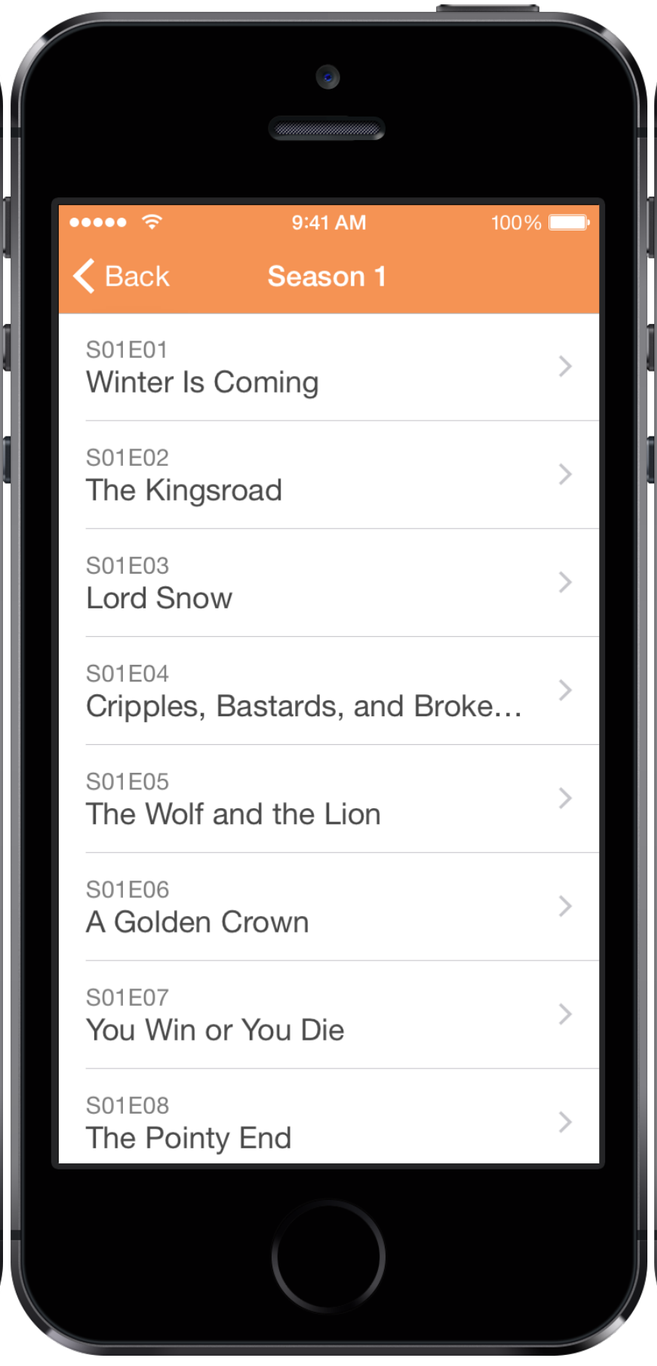
- Só chamar!
- marcelo.fabri@movile.com
- me@marcelofabri.com
- Grupo de email

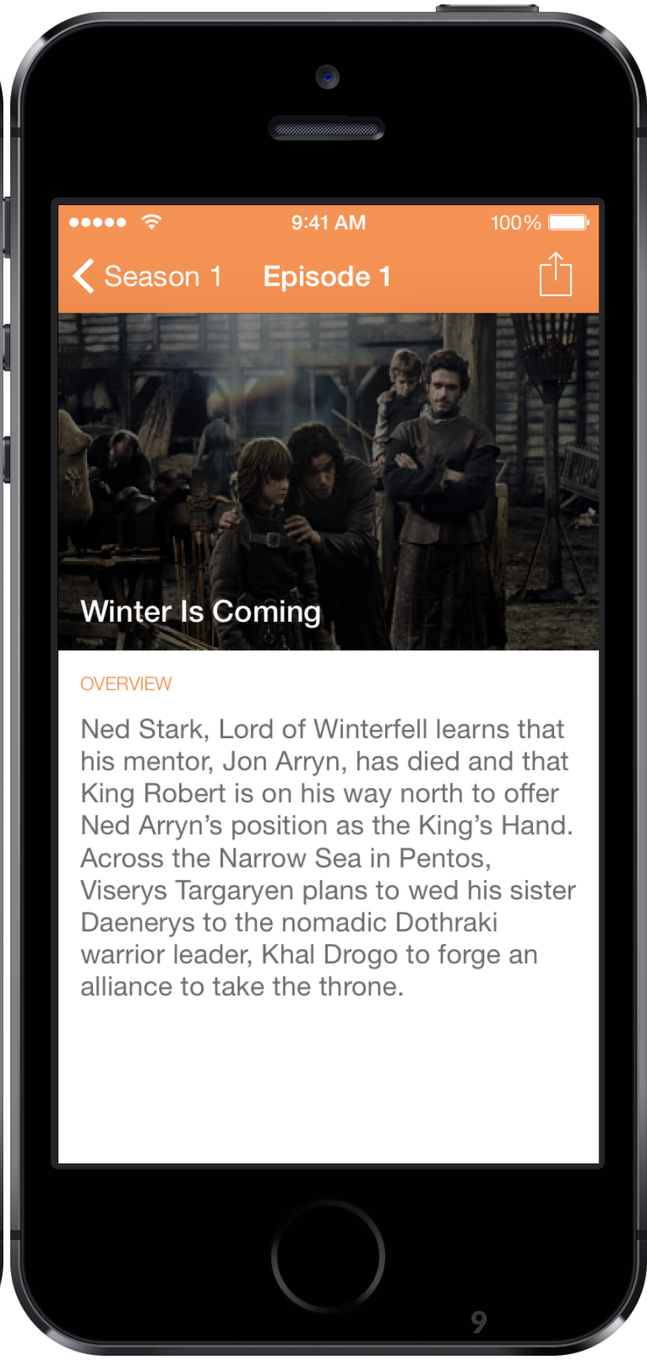
Projeto prático

- Listagem de episódios de séries
- Integração com o trakt.tv para obter os dados


9:41 AM
100%
Shows
Popular
Favorites
Band of Brothers
Planet Earth
Sherlock
House of Cards
Breaking Bad
Game of Thrones
Firefly
Fargo


9:41 AM
100%
Shows
Game of Thrones
9.4
STORYLINE
Seven noble families fight for control of the mythical land of Westeros. Friction between the houses leads to full-scale war. All while a very ancient evil awakens in the farthest north. Amidst the war, a neglected military order of misfits, the Night's Watch, is all that stands between the realms of men and the icy horrors beyond.
SEASONS
Season 5


9:41 AM
100%
Back
Season 1
S01E01
Winter Is Coming
S01E02
The Kingsroad
S01E03
Lord Snow
S01E04
Cripples, Bastards, and Broke…
S01E05
The Wolf and the Lion
S01E06
A Golden Crown
S01E07
You Win or You Die
S01E08
The Pointy End

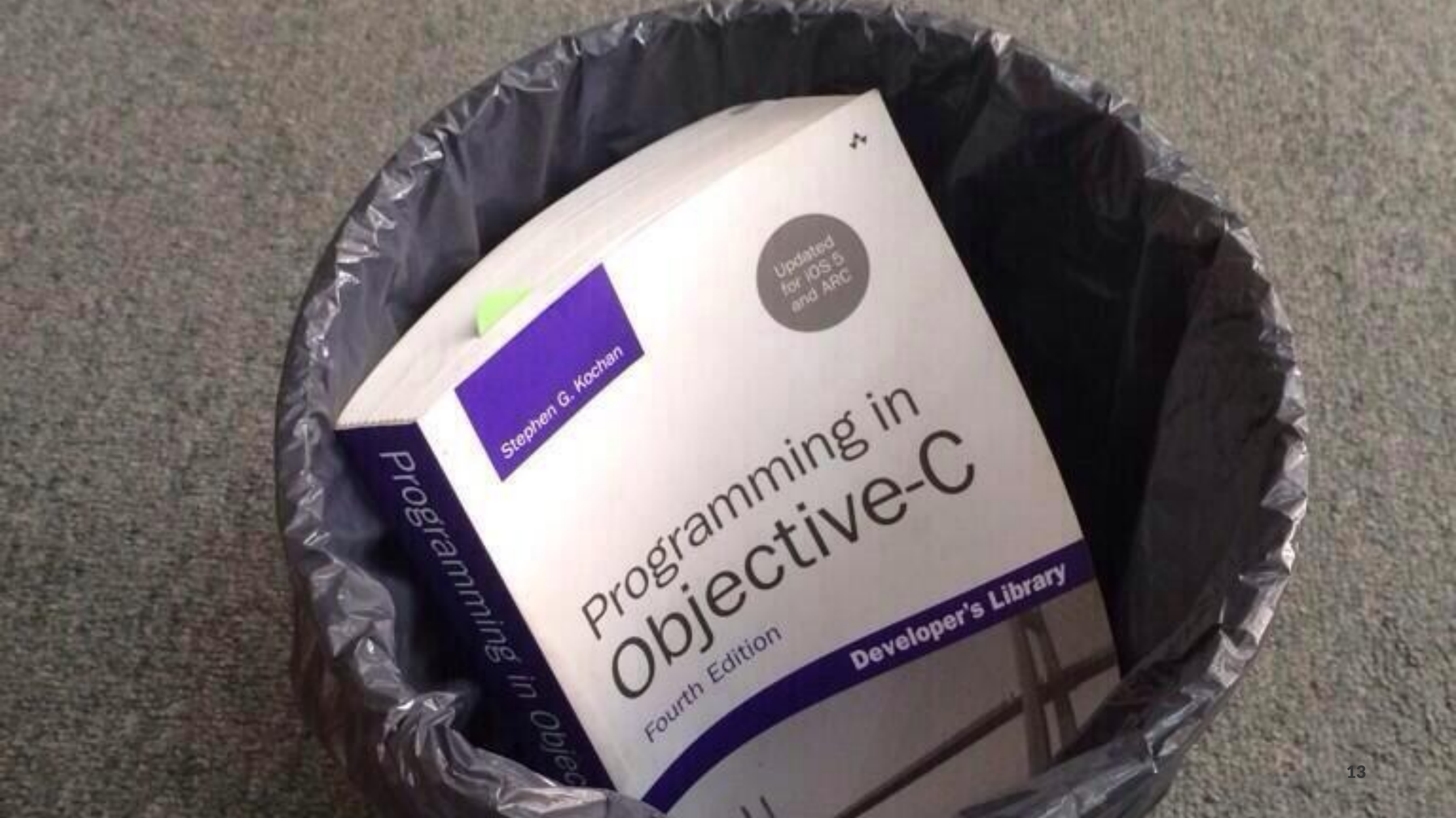

9:41 AM
100%
Season 1
Episode 1
Winter Is Coming
OVERVIEW
Ned Stark, Lord of Winterfell learns that his mentor, Jon Arryn, has died and that King Robert is on his way north to offer Ned Arryn's position as the King's Hand. Across the Narrow Sea in Pentos, Viserys Targaryen plans to wed his sister Daenerys to the nomadic Dothraki warrior leader, Khal Drogo to forge an alliance to take the throne.

"Bibliografia"

- Apple Developer Portal
- Ray Wenderlich
- NSHipster
- Little Bites of Cocoa
- Natasha The Robot
- iOS Dev Weekly
- objc.io

Swift x Objective-C

Swift x ~~Objective-C~~


Stephen G. Kochan
Updated for iOS 5 and ARC
Programming in Objective-C
Fourth Edition
Developer's Library
Programming in Objec

Chris Lattner
@clattner_llvm

Seguir

Looking forward to next month: I'll be the first and only guy with 4 years of swift programming experience :-)

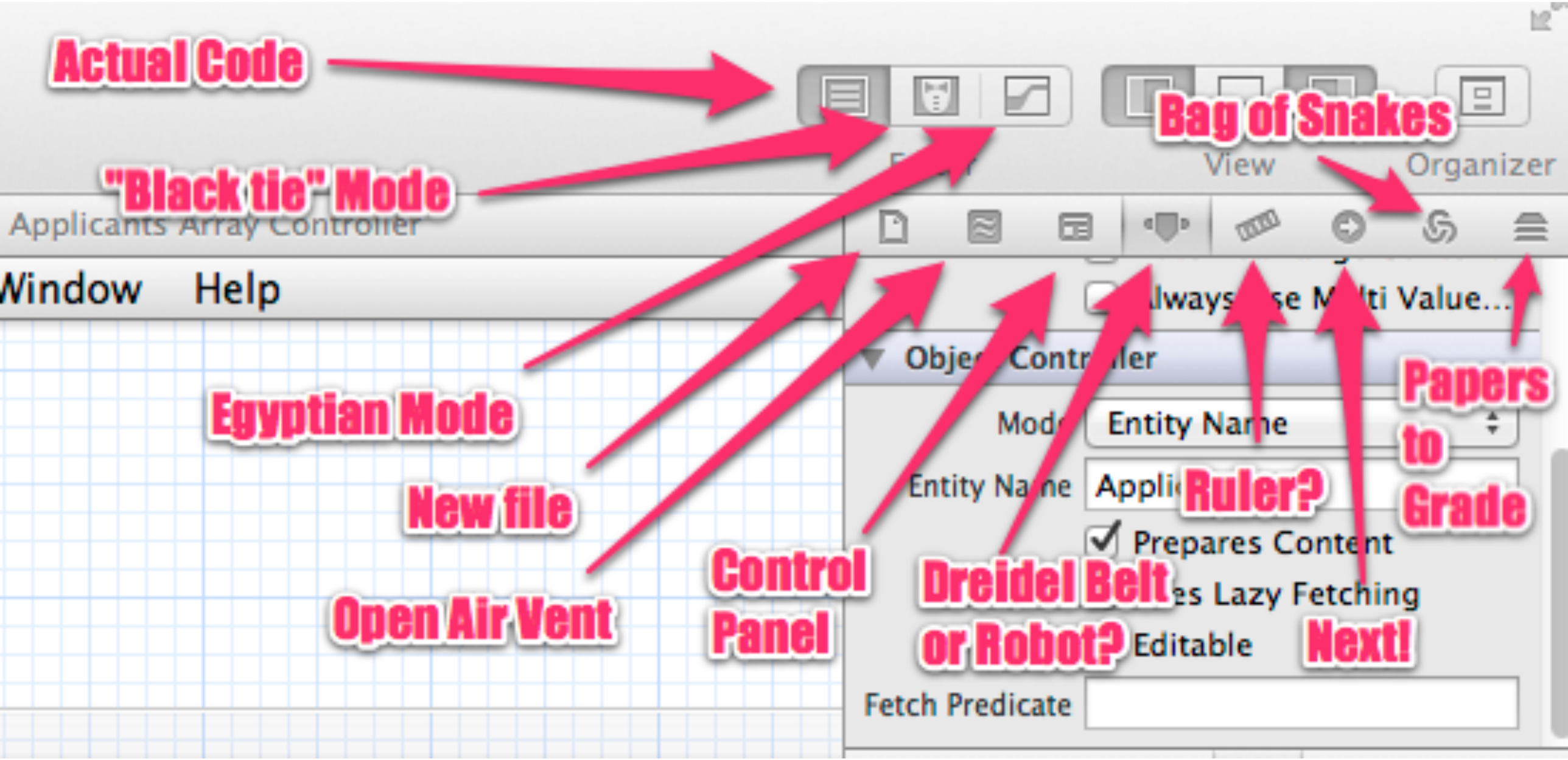
Responder Retweetar Curtir ••• Mais

RETWEETS
1.144

CURTIRAM
896

07:35 - 3 de jun de 2014


XCODE


Actual Code
"Black tie" Mode
Egyptian Mode
New file
Open Air Vent
Control Panel
Dreidel Belt or Robot?
Ruler?
Next!
Papers to Grade
Bag of Snakes
View
Organizer
Applicants Array Controller
Window
Help
Entity Name
Prepares Content
Editable
Fetch Predicate

Welcome to Xcode

Version 6.0 (6A215l)

Get started with a playground
Explore new ideas quickly and easily

Create a new Xcode project
Start building a new iPhone, iPad or Mac application.

Check out an existing project
Start working on something from an SCM repository.

☑ Show this window when Xcode launches

No Recent Projects

Open another project…


Swift

Functional patterns

Protocols and extensions on structs

Pattern matching

Concise syntax

Closures

Generics

Fast iteration

Native collections

Optional types

Operator overloading

Object orientation

Namespaces

Tuples

Type inference

Clear mutability syntax

Read-Eval-Print-Loop (REPL)

Interactive playground

Multiple return types

Compile to native code

Moderna

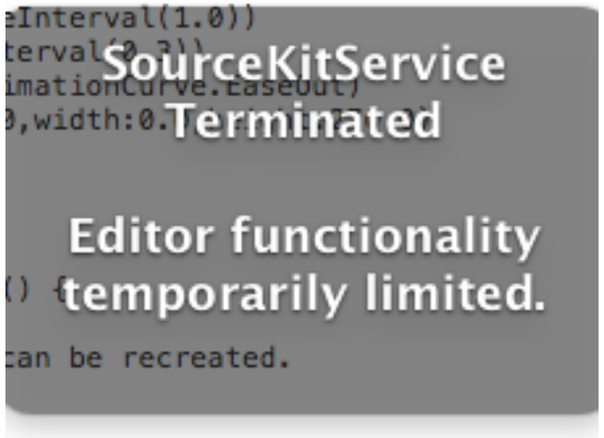
- Closures
- Tuplas
- Generics
- Structs/enums
- Características de programação funcional
- Sem colchetes, ponto-e-vírgula ou headers

Segura

- Variáveis sempre inicializadas antes de uso
- Tipagem forte
- Gerenciamento de memória automático (ARC)
- Erros em tempo de compilação, não de execução
- Inferência de tipos
- Conversão de tipos deve ser sempre explícita

Maaaaaaas...

- Ferramentas instáveis
- Xcode, clang, SourceKit, frameworks, etc
- Preparem-se para ficar um pouco estressados 😬



Sintaxe

Inspirado **fortemente** em "A Swift Tour"[1] e "Language Guide"[2]

---

[1] https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/GuidedTour.html

[2] https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Hello, world!

```
println("Hello, world!")
```

Variáveis

```
var myVariable = 42
myVariable = 50
let myConstant = 42

let orangesAreOrange = true
let turnipsAreDelicious = false
```

Observe a inferência de tipos!

Tipos explícitos

```
let implicitInteger = 70
let implicitDouble = 70.0
let explicitDouble: Double = 70
```

Conversão de tipos sempre explícita

```
let label = "The width is "                    // String
let width = 94                                 // Int
let widthLabel = label + String(width)         // String
```

Vetores

```
var shoppingList = ["catfish", "water", "tulips", "blue paint"]
shoppingList[1] = "bottle of water"
```

Dicionários

```
var occupations = [
    "Malcolm": "Captain",
    "Kaylee": "Mechanic",
]
occupations["Jayne"] = "Public Relations"
```

Controle de fluxo

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
println(teamScore)
```

Chaves sempre obrigatórias
Parênteses opcionais

for vs for-in

```
var firstForLoop = 0
for i in 0..<4 {
    firstForLoop += i
}
println(firstForLoop)

var secondForLoop = 0
for var i = 0; i < 4; ++i {
    secondForLoop += i
}
println(secondForLoop)
```

while vs do-while

```
var n = 2
while n < 100 {
    n = n * 2
}
println(n)

var m = 2
do {
    m = m * 2
} while m < 100
println(m)
```

switch

```
let vegetable = "red pepper"
switch vegetable {
case "celery":
    let vegetableComment = "Add some raisins and make ants on a log."
case "cucumber", "watercress":
    let vegetableComment = "That would make a good tea sandwich."
case let x where x.hasSuffix("pepper"):
    let vegetableComment = "Is it a spicy \(x)?"
default:
    let vegetableComment = "Everything tastes good in soup."
}
```

Não tem `break` (igual em C).
Se quiser continuar, use `fallthrough`.

Tuplas

```
let http404Error = (404, "Not Found") // (Int, String)
let (statusCode, statusMessage) = http404Error

println("The status code is \(statusCode)")
println("The status message is \(statusMessage)")



println("The status code is \(http404Error.0)")
println("The status message is \(http404Error.1)")

// Só queremos o status code
let (justTheStatusCode, _) = http404Error
```

Optionals

- Se um valor pode ser nulo, ele tem que ser anotado de forma diferente

```
let possibleNumber = "123"
let convertedNumber = possibleNumber.toInt()
```

- `convertedNumber` é do tipo `Int?`, já que `possibleNumber` pode não ser um inteiro
- Qualquer tipo pode ter sua variante optional

Forced Unwrapping

```
if convertedNumber != nil {
    println("Value == \(convertedNumber!).")
}
```

Se `convertedNumber` fosse `nil`, um erro (irrecuperável) em runtime seria disparado.

Optional Binding

```
if let actualNumber = possibleNumber.toInt() {
    println("Value == \(actualNumber)")
} else {
    println("Not an integer")
}

// Multiple optional binding
if let  constantName = someOptional,
        anotherConstantName = someOtherOptional {

}
```

Implicitly Unwrapped Optionals

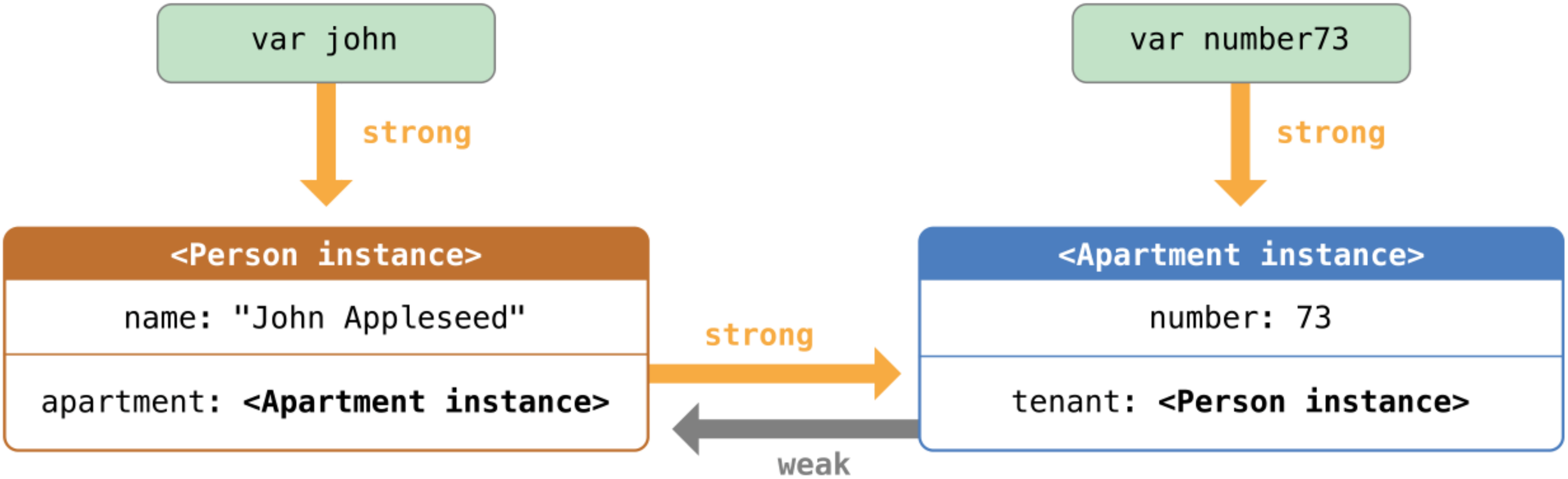
- Todas variáveis de um objeto devem estar instanciadas ao fim de seu construtor
- Mas às vezes só conseguimos fazer isso em algum momento posterior
    - Exemplo: carregamento de telas

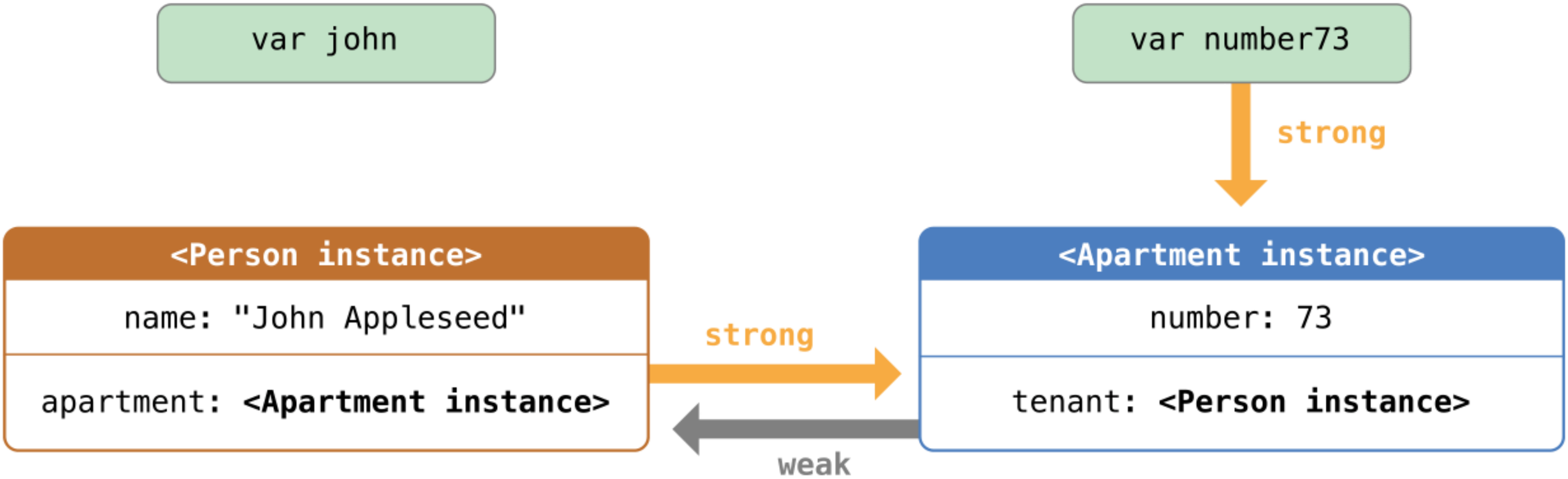
```
@IBOutlet weak var label: UILabel!
```

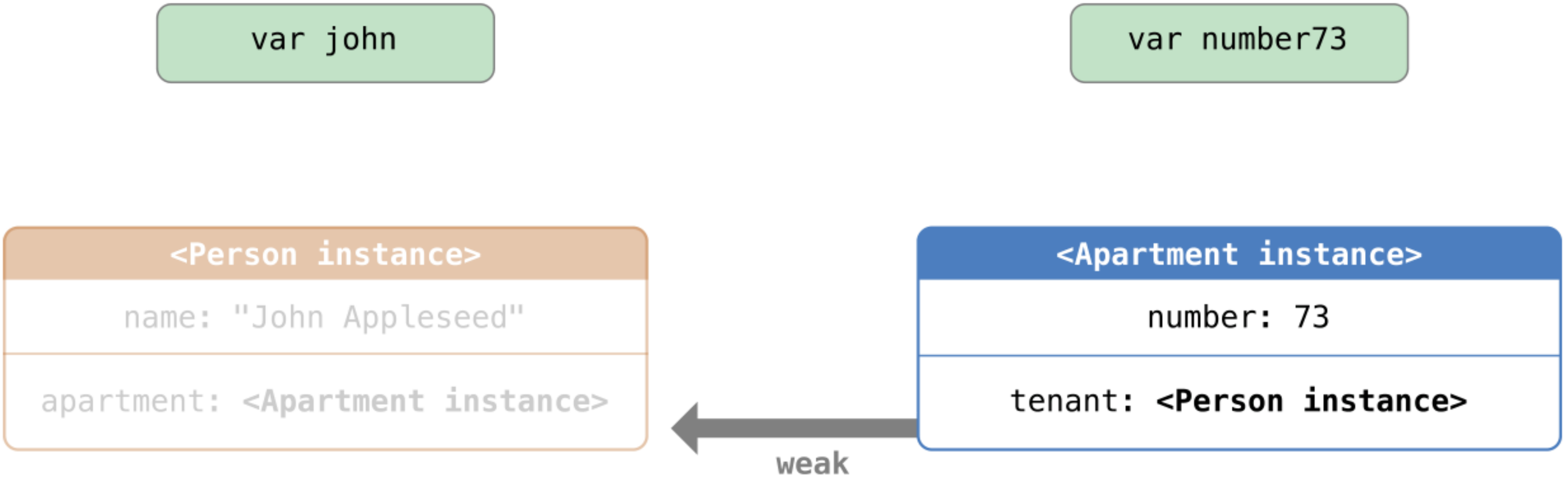
weak vs strong

```
class Person {
    let name: String
    init(name: String) { self.name = name }
    var apartment: Apartment?
    deinit { println("\(name) is being deinitialized") }
}

class Apartment {
    let number: Int
    init(number: Int) { self.number = number }
    weak var tenant: Person?
    deinit { println("Apartment #\(number) is being deinitialized") }
}
```

```
var john: Person?
var number73: Apartment?

john = Person(name: "John Appleseed")
number73 = Apartment(number: 73)

john!.apartment = number73
number73!.tenant = john



john = nil
// prints "John Appleseed is being deinitialized"

println(number73!.apartment) // "nil"
```


var john
strong
var number73
strong
<Person instance>
name: "John Appleseed"
apartment: <Apartment instance>
strong
weak
<Apartment instance>
number: 73
tenant: <Person instance>


var john
var number73
strong
<Person instance>
name: "John Appleseed"
apartment: <Apartment instance>
strong
weak
<Apartment instance>
number: 73
tenant: <Person instance>


var john
var number73
<Person instance>
name: "John Appleseed"
apartment: <Apartment instance>
<Apartment instance>
number: 73
tenant: <Person instance>
weak

`weak` vs `strong`

- `strong` é o padrão (implícito)
- Veremos mais sobre gerenciamento de memória mais tarde
- Em geral, properties de elementos de tela são `weak` (a hierarquia de view impede que seja desalocadas imediatamente)

```
@IBOutlet weak var label: UILabel!
```

Optional Chaining

```
if let beginsWithThe =
    john.residence?.address?.buildingIdentifier()?.hasPrefix("The") {
        if beginsWithThe {
            println("John's building identifier begins with \"The\".")
        } else {
            println("John's building identifier does not begin with \"The\".")
        }
}
```

Só imprimirá algo se `residence`, `address` e `buildingIdentifier()` forem diferente de `nil`.

`protocol`

- Definição de interface, não implementação
- Diminuição de acoplamento
- Similar ao `interface` de Java

```
protocol Named {
    var name: String { get set }

    func printName() -> Void
}
```

Lição de casa: ver como tornar um método opcional.

`struct` vs `class`

- Guardar informações
- Métodos
- Construtores (`init`)
- Extensions
- Protocols

`class`

- Herança
- Destrutores (`deinitializer`)
- Contagem de referência: passagem por referência
- *reference type*

`struct`

- Passagem por valor (ou seja, sempre são copiadas)
- *value type*

```
class PersonClass: Named {
    var name: String = ""

    init(name: String) {
        self.name = name
    }

    func printName() {
        println("I'm named \(name) and I'm a class.")
    }
}

struct PersonStruct: Named {
    var name: String = ""

    func printName() {
        println("I'm named \(name) and I'm a struct.")
    }
}
```

```swift
func changeName(var obj: Named) {
    let oldName = obj.name
    obj.name = "Someone Else"
    println("Just changed my name from \(oldName) to \(obj.name).")
}

var p1 = PersonClass(name: "Marcelo")
p1.printName() // I'm named Marcelo and I'm a class.
changeName(p1) // Just changed my name from Marcelo to Someone Else.
p1.printName() // I'm named Someone Else and I'm a class.

var p2 = PersonStruct(name: "John")
p2.printName() // I'm named John and I'm a struct.
changeName(p2) // Just changed my name from John to Someone Else.
p2.printName() // I'm named John and I'm a struct.
```

http://stackoverflow.com/questions/27241411/cannot-assign-to-property-in-protocol-swift-compiler-error

enum

```
enum ShowStatus: String {
    case Returning = "returning series"
    case InProduction = "in production"
    case Canceled = "canceled"
    case Ended = "ended"

    func isFinished() -> Bool {
        switch self {
        case .Returning, .InProduction:
            return false
        case .Canceled, .Ended:
            return true
        }
    }
}
```

Usando enum

```
let returning = ShowStatus.Returning
if let returningFromString = ShowStatus(rawValue: "returning series") {
    println("Yey!")
}

println(ShowStatus.Ended.rawValue) // "ended"
```

enum com valores associados

```
enum Router {
    case PopularShows
    case Show(String)
    case Seasons(showId: String)
    case Episodes(showId: String, season: Int)
    case Episode(showId: String, season: Int, number: Int)
}

let popular = Router.PopularShows
let got = Router.Show("game-of-thrones")
let gotSeasons = Router.Seasons(showId: "game-of-thrones")
```

```swift
extension Router {
    var path: String {
        switch self {
        case .PopularShows:
            return "shows/popular"
        case .Show(let id):
            return "shows/\(id)"
        case .Seasons(let showId):
            return "shows/\(showId)/seasons"
        case .Episodes(let showId, let season):
            return "shows/\(showId)/seasons/\(season)"
        case .Episode(let showId, let season, let number):
            return "shows/\(showId)/seasons/\(season)/episodes/\(number)"
        }
    }
}

Router.PopularShows.path              // "shows/popular"
Router.Show("game-of-thrones").path // "shows/game-of-thrones"
```

`extension`

- Adicionar funcionalidade a um tipo (`class`, `struct` ou `enum`)
- Não precisa ter o código fonte original
- Não consegue sobrescrever funcionalidades existentes
- Pode ser usada para fazer um tipo implementar um `protocol`

Exemplo de `extension`

```
extension Int {
    mutating func square() {
        self = self * self
    }

    func half() -> Int {
        return self / 2
    }
}

var someInt = 4
someInt.square() // someInt is now 16

var anotherInt = someInt.half()
// anotherInt is 8, but someInt is 16
```

Melhores práticas/convenções

http://ericasadun.com/2015/05/05/swift-dont-do-that/

- Don’t fight type inference.
- Don’t use var when let is appropriate.
- Don’t use classes when structs will do.
- Do use Swift native types where possible.
- Do prefer if-let to forced unwrapping whenever possible.
- Don’t add code cruft.

Exercício

Criar modelo Show que será usado pelo app

Dica: criar modelos auxiliares para os ids e URLs de imagens

Documentação do Trakt.tv: http://docs.trakt.apiary.io