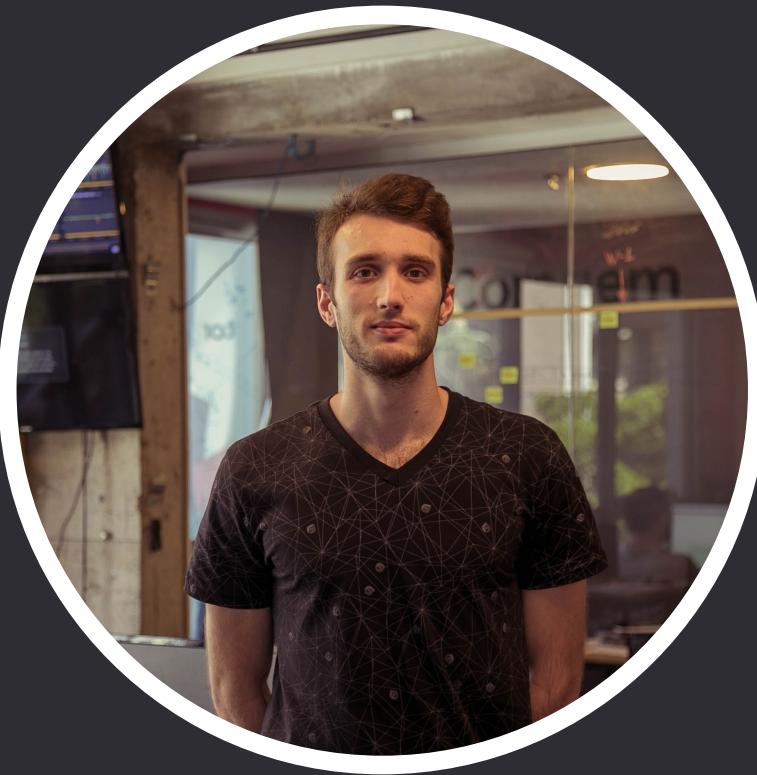


5 COISAS

que toda pessoa desenvolvedora iOS deveria saber



Txai Wieser

Ciência da Computação @ UFRGS

Apple Developer Academy

Coordenador da trilha iOS do TDC

Mobile Lead Developer @ Warren



warren

5 COISAS

que toda pessoa desenvolvedora iOS deveria saber

Informações Sensíveis do App

```
static let url = "https://dev-api.olivarrer.com"  
static let apiKey = "keyyy03499tchbz1d09832kjl#65alkd"  
  
→ SecurityExample.app strings SecurityExample  
Data: 2009  
https://dev-api.olivarrer.com  
keyyy03499tchbz1d09832kjl#65alkd  
1cc29046e848a304a851a0332a057f  
33200#65alkd  
window.  
window.
```

Informações Sensíveis do App

```
// Configuração Falsa do App  
let o = Obfuscator.withDefaults([AppDelegate.self])  
let apiKey = o.copyWithString(string: apiKey)  
o.setStrings(["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "aa", "bb", "cc", "dd", "ee", "ff", "gg", "hh", "ii", "jj", "kk", "ll", "mm", "nn", "oo", "pp", "qq", "rr", "ss", "tt", "uu", "vv", "ww", "xx", "yy", "zz"],  
    forKey: "apiKeys")  
  
// Acesso no App  
let apiKeyObjs: [UInt8] = [48, 51, ..., 54, 52]  
let original = o.copyWith(key: apiKeyObjs)
```

Extras

- Se seu app depende muito de segurança existem diversas técnicas, mas custam cara e exigem tempo.
- Debug e jailbreak detection pode ser uma opção.
- Obscuração total do código.
- Não existe maneira fácil

Dicas de Segurança



Txaí Wieser

linkedin.com/in/bawieser
github.com/bawieser
twitter.com/bawieser
bawieser@cloud.com

170 slides

em

45

minutos?



CHALLENGE ACCEPTED

PRIMEIRA

Swift Recap

Swift Recap

- Value Types vs. Reference Types
- Optionals
- Closures

Funções de Alta Ordem

Funções de Alta Ordem

- Map
- Filter
- Reduce.

Gerenciamento de Memória

Gerenciamento de Memória

- **Modelo de referencias**
- **Ciclos de referencias**
- **Strong, Weak e Unowned**

Técnicas de DEBUG

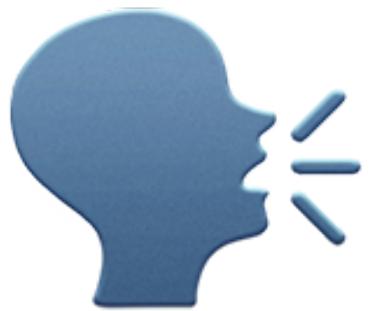
Técnicas de DEBUG

- LLDB
- Breakpoints

Dicas de Segurança

Dicas de Segurança

- Armazenando Dados
- Networking
- Informações Sensíveis



Sintaxe básica
Control Flow



Swift Recap

- **Value Types vs. Reference Types**
- **Optionals**
- **Closures**

Swift Recap

* Value Types vs. Reference Types

- Optionals
- Closures

Value Types

Referencias guardam uma copia **única** de seus dados.

Reference Types

Referencias compartilham uma **única** cópia de seus dados.

Value Types

Representado por **classes**

Reference Types

Representado por **structs,
enums, tuplas, primitivos**

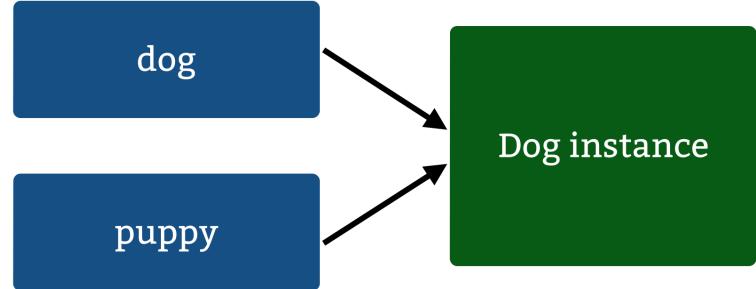
Reference Types

```
class Dog {  
    var wasFed = false  
}
```

```
let dog = Dog()
```

```
let puppy = dog  
puppy.wasFed = true
```

```
dog.wasFed      // true  
puppy.wasFed   // true
```



Value Types

```
var a = 42
```

```
var b = a
```

```
b+=1
```

```
a    // 42  
b    // 43
```

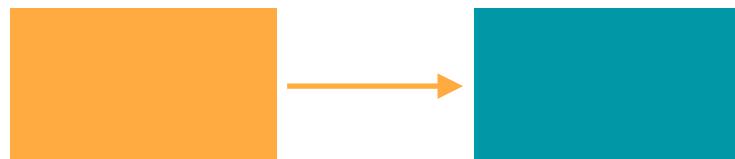
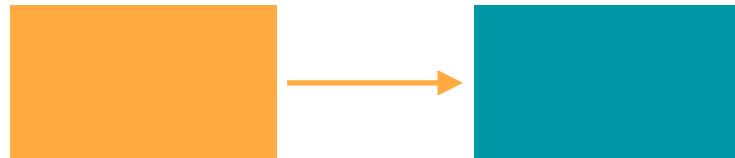
Value Types

```
struct Cat {  
    var wasFed = false  
}
```

```
var cat = Cat()
```

```
var kitty = cat  
kitty.wasFed = true
```

```
cat.wasFed           // false  
kitty.wasFed        // true
```



Mutabilidade

let

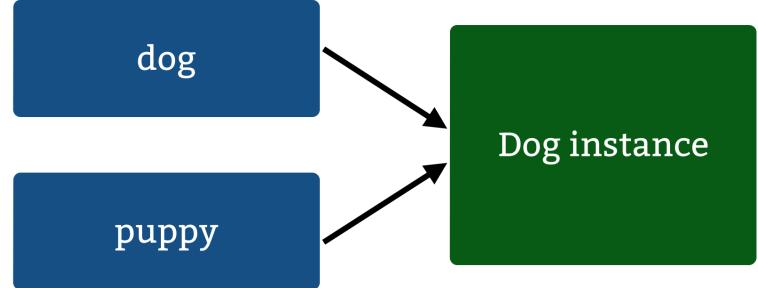
Constante

var

Variável

Reference Types

```
class Dog {  
    var wasFed = false  
}  
  
let dog = Dog()  
  
let puppy = dog  
puppy.wasFed = true  
  
dog.wasFed      // true  
puppy.wasFed    // true
```



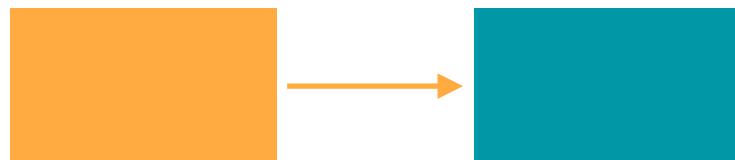
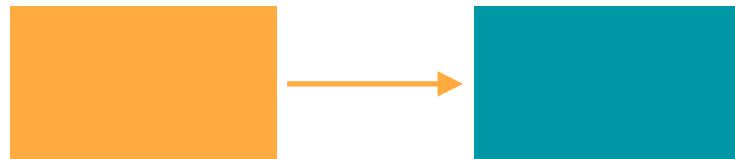
Value Types

```
struct Cat {  
    var wasFed = false  
}
```

```
let cat = Cat()
```

```
var kitty = cat  
kitty.wasFed = true
```

```
cat.wasFed           // false  
kitty.wasFed        // true
```



**E agora, qual devo
escolher?**

IDENTITY vs EQUALITY

Quando usar Value Type?

- Onde comparar instancias com == faz sentido
- Os dados vão ser usando no código em multiplas threads

```
struct Point {  
    var x: Float  
    var y: Float  
}  
  
let point1 = Point(x: 2, y: 3)  
let point2 = Point(x: 2, y: 3)  
  
if point1 == point2 {  
    // São iguais  
}
```

==

Faz
Sentido

```
struct Point {  
    var x: Float  
    var y: Float  
}  
  
struct Shape {  
    var center: Point  
}
```

```
let initialPoint = Point(x: 100, y: 200)
```

```
var circle = Shape(center: initialPoint)  
var square = Shape(center: initialPoint)
```

```
square.center.x = 200  
circle.center // A posição  
//de circle deveria mudar também?
```

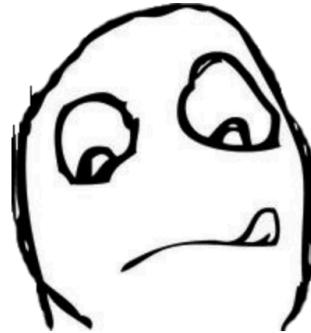
Copias Independentes

Quando usar Reference Type?

- Onde comparar instâncias com === faz sentido
- Compartilhamento de estado

```
class Account {  
    var balance = 0.0  
}  
  
class Person {  
    let account: Account  
    init(_ account: Account) { self.account = account }  
}  
  
let account = Account()  
let person1 = Person(account)  
let person2 = Person(account)  
person2.account.balance += 100.0  
  
print(person1.account.balance)      // 100  
print(person2.account.balance)      // 100
```

class? struct?



struct!

(at least to start...)

Queremos imutabilidade

Swift

structs: 124, enums: 19, classes: 3

Objective-C

NSString vs. NSMutableString e NSArray vs. NSMutableArray.

Copy-on-Write

Swift implementa nativamente Value Types com Copy-on-Write.

- Mesma eficiência de passagem por referência.
- Segurança da imutabilidade quando necessário.

Swift Recap

Value Types vs. Reference Types

- Optionals
- Closures

Swift Recap

✓ Value Types vs. Reference Types

* Optionals

- Closures

Optionals

[...] tipos opcionais lidam com a ausência de um valor. Eles dizem “há um valor e ele é x” ou “não há valor algum”. Usá-los é semelhante a usar nil com ponteiros em Objective-C mas eles funcionam para todos os tipos, não só para classes. Eles não só são mais seguros e expressivos do que ponteiros nulos, como estão no coração de muitas das funcionalidades mais poderosas de Swift.

```
struct Caixa {  
    let presente  
}
```

```
let presenteAmigoSecreto: Caixa
```

```
enum Caixa {  
    case algumaCoisa  
    case nada  
}
```

```
let presente: Caixa  
  
switch presente {  
    case .algumaCoisa:  
        print("OBA!")  
    case .nada:  
        print(":(")  
}
```

```
enum Caixa {
    case algumaCoisa(Bicicleta)
    case nada
}
```

```
switch presente {
case .algumaCoisa(let bike):
    print("OBA! Uma bicicleta da \\" + bike.marca + ")"
case .nada:
    print(":(")
}
```

```
struct Bicicleta {
    let marca: String
}
```

```
enum Caixa {
    case algumaCoisa(Any)
    case nada
}

switch presente {
    case .algumaCoisa(let algo):
        print("OBA! ??????????")
    case .nada:
        print(":(")
}
```

```
enum Optional<T> {
    case some(T)
    case none

    init() { self = .none }
    init(_ some: T) { self = .some(some) }
}
```

```
let presente = Optional<Patinete>()
let presente = Optional<Patinete>(patinete)
```

```
let presente: Optional<Patinete> = .none
let presente: Optional<Patinete> = .some(patinete)
```

```
enum Optional<T> {
    case some(T)
    case none

    init() { self = .none }
    init(_ some: T) { self = .some(some) }
}
```

```
let presente = Optional<Patinete>()
let presente = Optional<Patinete>(patinete)
```

```
let presente: Optional<Patinete> = .none
let presente: Optional<Patinete> = .some(patinete)
```

```
let presente: Patinete? = .none
let presente: Patinete? = .some(patinete)
let presente: Patinete? = nil
let presente: Patinete? = patinete
```

```
let numeroSorteio1: Int? = nil  
let numeroSorteio2: Int = 5  
  
let soma = numeroSorteio1 + numeroSorteio2
```

```
let numeroSorteio1: Int? = nil
let numeroSorteio2: Int = 5

let soma = numeroSorteio1 + numeroSorteio2

switch numeroSorteio1 {
    case .none:
        let soma = numeroSorteio2

    case .some(let numeroSorteio1):
        let soma = numeroSorteio1 + numeroSorteio2
}

if let numeroSorteio1 = numeroSorteio1 {
    let soma = numeroSorteio1 + numeroSorteio2
} else {
    let soma = numeroSorteio2
}
```

EVERY TIME YOU USE !



A KITTEN DIES

Swift Recap

✓ Value Types vs. Reference Types

✓ Optionals

- Closures

Swift Recap

✓ Value Types vs. Reference Types

✓ Optionals

* Closures

Closures

Closures são blocos de funcionalidade autocontidos que podem ser passados adiante e usados no seu código.

– Apple

Closures

Funções Globais

Funções aninhadas

Expressões Closure

Closures are headless
functions

Closures are headless functions

ou seja, sem a keyword **func** e sem o **nome**

```
{ ( parameters ) -> return type in  
    statements  
}
```

```
let sayHello = { (name: String) -> String in
    return "Hello \\" + name + "\""
}

sayHello("Txai") // "Hello Txai"
```

(String) -> String

```
let sayHello = { (name: String) -> String in
    return "Hello \\" + name + "\""
}
```

(String) -> String

```
let sayHello: ((String) -> String) = { (name: String) -> String in
    return "Hello \\" + name + "\""
}
```

(String) -> String)

```
let sayHello: ((String) -> String) = { name in
    return "Hello \ (name)"
}
```

(String) -> String)

```
let sayHello: ((String) -> String) = { name in
    "Hello \ (name)"
}
```

(String) -> String

```
let sayHello: ((String) -> String) = { name in "Hello \\" + (name) "
```

(String) -> String

```
let sayHello: ((String) -> String) = { name in "Hello \$(name)" }
```

\$0, \$1, \$2 ...

(String) -> String

```
let sayHello: ((String) -> String) = { "Hello \($0)" }
```

Swift Recap

✓ Value Types vs. Reference Types

✓ Optionals

✓ Closures



Swift Recap

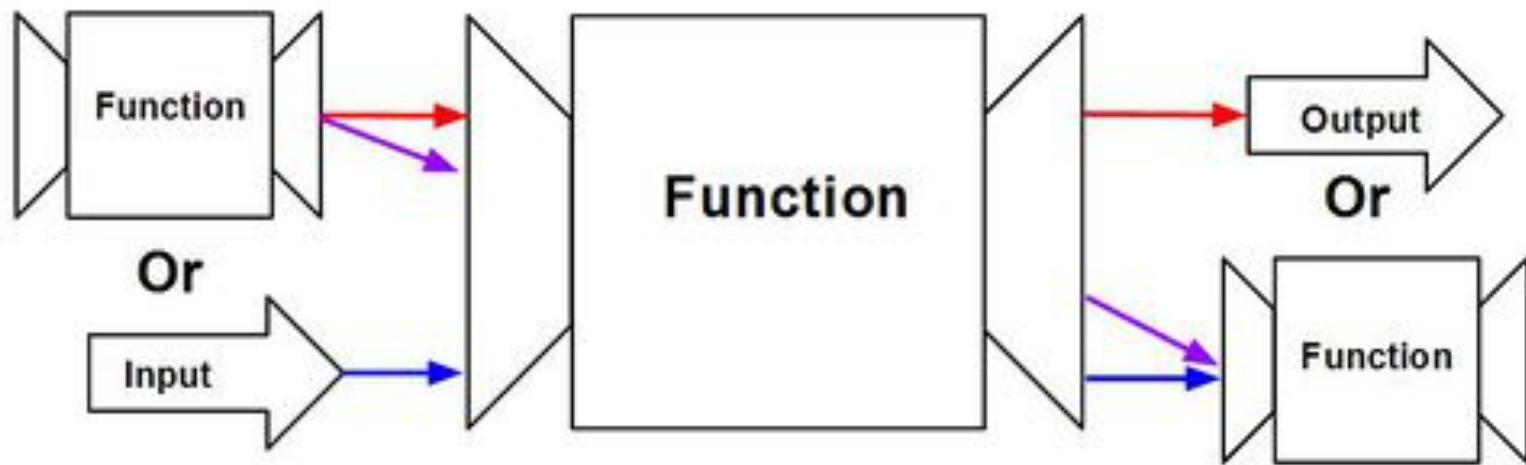
Funções de Alta Ordem

Funções de Alta Ordem

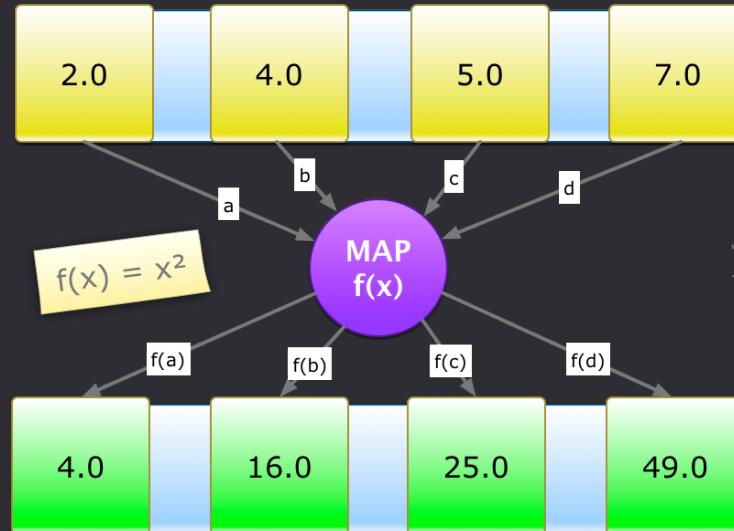
- Map
- Filter
- Reduce.

O que é uma função de alta ordem?

- Recebe uma função como parâmetro.
- Produz uma função como resultado.



Map



```
var alunosNotas = [4, 5, 7, 9, 6, 10, 3]

var novaNotas: [Int] = []

for nota in alunosNotas {
    let novaNota = nota + 1
    novaNotas.append(novaNota)
}

// Funcional

alunosNotas = alunosNotas.map( { (nota: Int) -> Int in
    return nota + 1
})
```

```
var alunosNotas = [4, 5, 7, 9, 6, 10, 3]

var novaNotas: [Int] = []

for nota in alunosNotas {
    let novaNota = nota + 1
    novaNotas.append(novaNota)
}

// Funcional

alunosNotas = alunosNotas.map( { (nota: Int) -> Int in
    return nota + 1
})
```

```
alunosNotas = alunosNotas.map( { (nota: Int) -> Int in
    return nota + 1
} )
```

```
alunosNotas = alunosNotas.map( { nota in
    return nota + 1
} )
```

```
alunosNotas = alunosNotas.map( { nota in
    nota + 1
} )
```

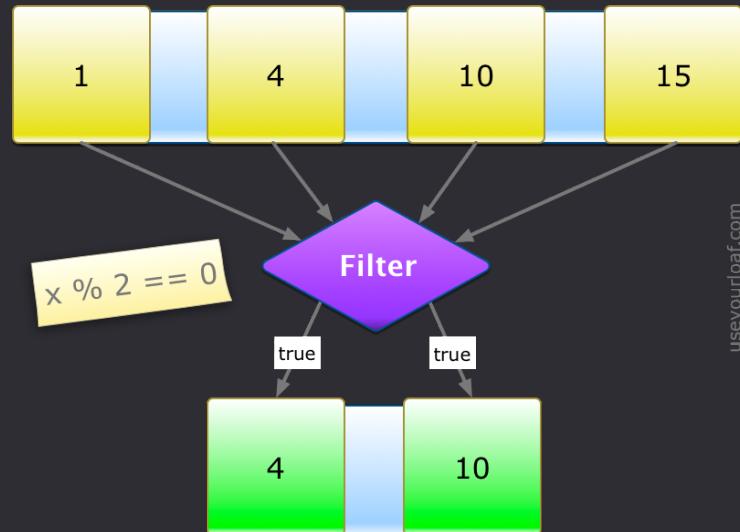
```
alunosNotas = alunosNotas.map( { nota in
    nota + 1
} )
```

```
alunosNotas = alunosNotas.map( { nota in nota + 1 } )
```

```
alunosNotas = alunosNotas.map( { $0 + 1 } )
```

```
alunosNotas = alunosNotas.map { $0 + 1 }
```

Filter



useyourloaf.com

```
var alunosNotas = [4, 5, 7, 9, 6, 10, 3]
```

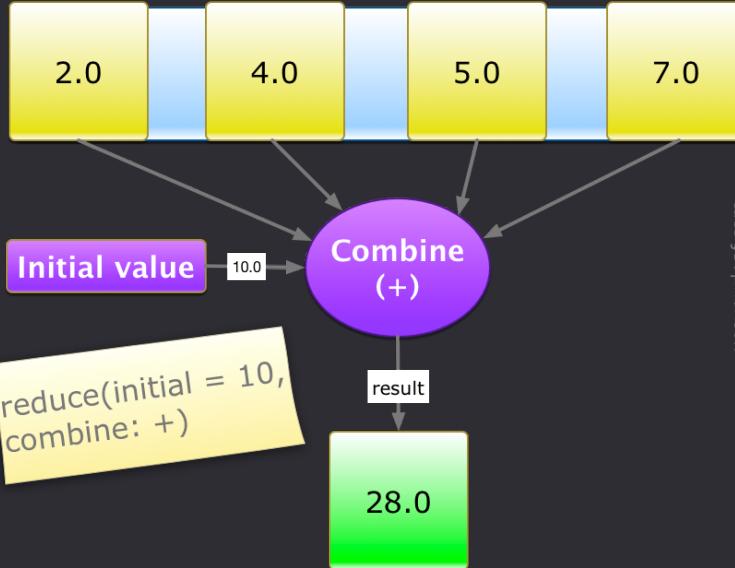
```
var aprovadosNotas: [Int] = []
```

```
for nota in alunosNotas {  
    if nota >= 5 {  
        aprovadosNotas.append(novaNota)  
    }  
}
```

// Funcional

```
var alunosNotas = [4, 5, 7, 9, 6, 10, 3]  
alunosNotas = alunosNotas.filter { $0 >= 5 }
```

Reduce



```
var alunosNotasReduce = [4, 5, 7, 9, 6, 10, 3]
```

```
var soma = 0
```

```
for nota in alunosNotasFilter {  
    soma += nota  
}
```

// Funcional

```
soma = alunosNotas.reduce(0, { $0 + $1 })  
soma = alunosNotas.reduce(0, +)
```

Encadeamento

```
let despesas = [4.50, 5.00, 8.30, 2.80, 9.10, 7.70]

let total = despesas.filter { $0 >= 7 }
    .map { $0 * 1.10 }
    .reduce(0, +)
```



Funções de Alta Ordem

Gerenciamento de Memória

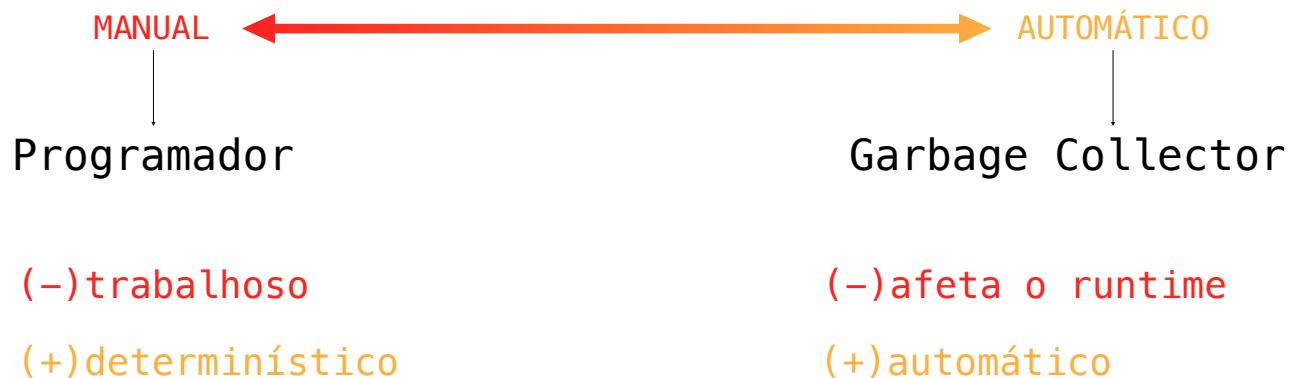
Gerenciamento de Memória

- **Modelo de referencias**
- **Ciclos de referencias**
- **Strong, Weak e Unowned**

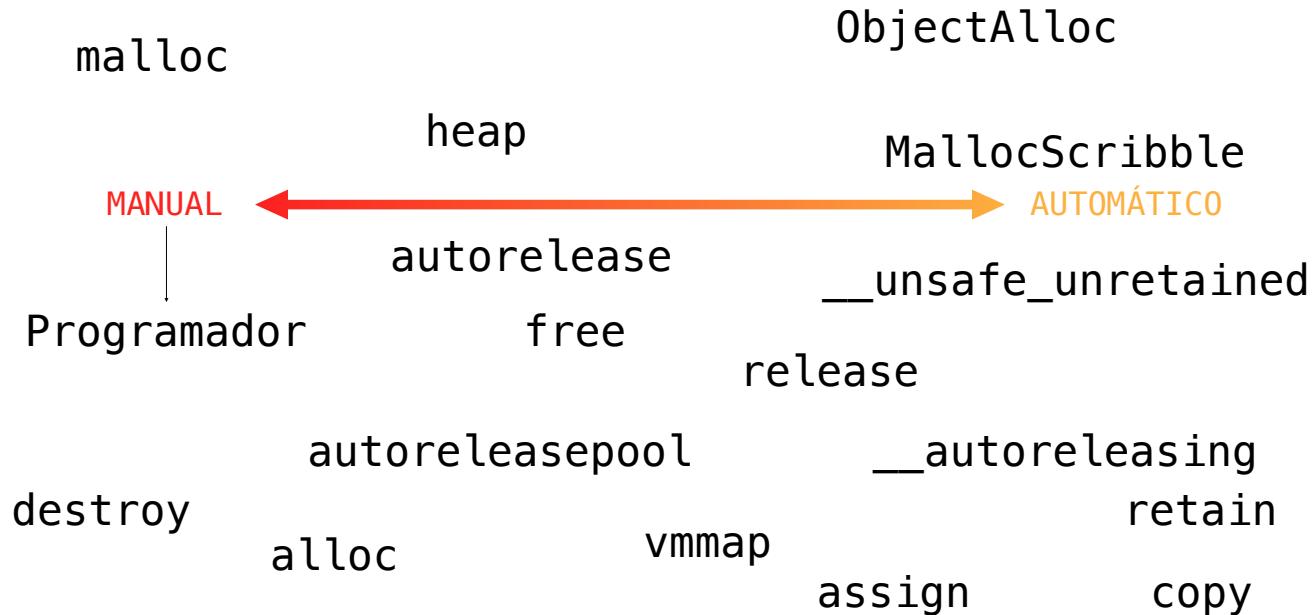
Gerenciador de Memória



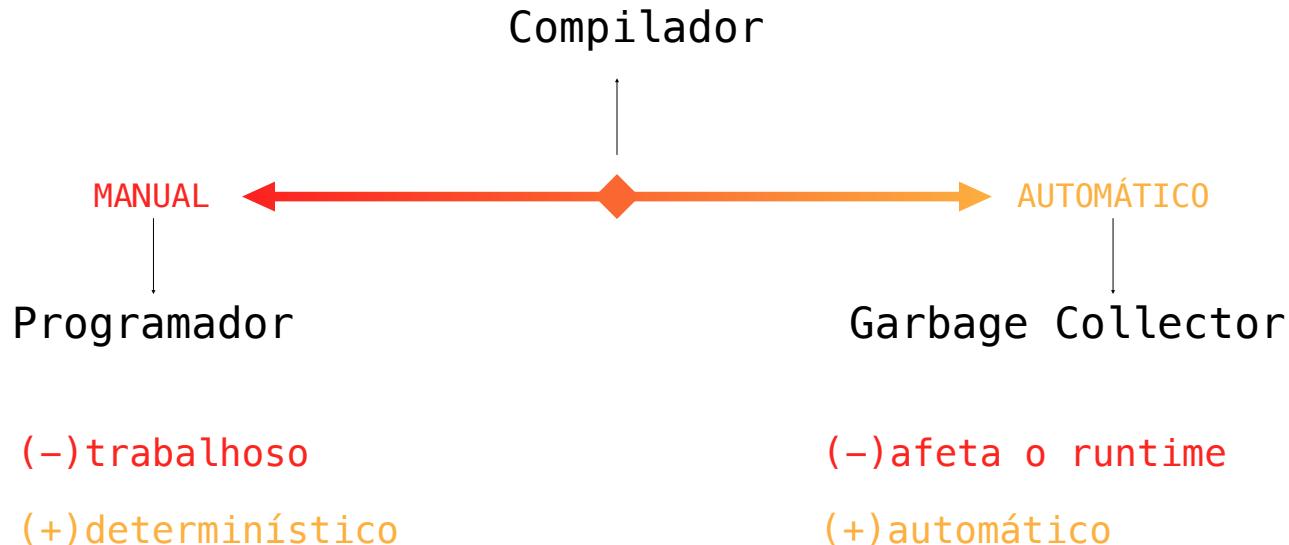
Gerenciador de Memória



Gerenciador de Memória



Gerenciador de Memória



ARC

Automatic Reference Counting

Automatic Reference Counting

strong

```
class Pessoa { }
```

```
var ref1: Pessoa?
```

ref1

nil

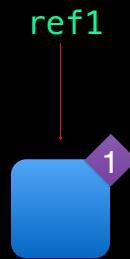
Automatic Reference Counting

strong

```
class Pessoa { }

var ref1: Pessoa?

ref1 = Pessoa()
```



Automatic Reference Counting

strong

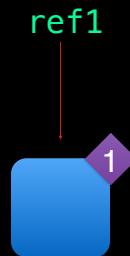
```
class Pessoa { }
```

```
var ref1: Pessoa?
```

```
ref1 = Pessoa()
```

```
var ref2: Pessoa?
```

ref2



Automatic Reference Counting

strong

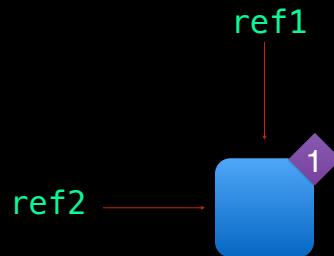
```
class Pessoa { }
```

```
var ref1: Pessoa?
```

```
ref1 = Pessoa()
```

```
var ref2: Pessoa?
```

```
ref2 = ref1
```



Automatic Reference Counting

strong

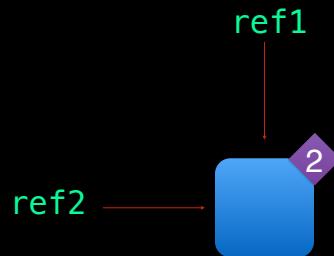
```
class Pessoa { }
```

```
var ref1: Pessoa?
```

```
ref1 = Pessoa()
```

```
var ref2: Pessoa?
```

```
ref2 = ref1
```



Automatic Reference Counting

strong

```
class Pessoa { }
```

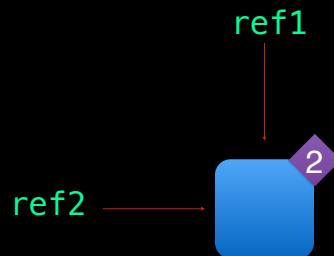
```
var ref1: Pessoa?
```

```
ref1 = Pessoa()
```

```
var ref2: Pessoa?
```

```
ref2 = ref1
```

```
ref1 = nil
```



Automatic Reference Counting

strong

```
class Pessoa { }
```

ref1

```
var ref1: Pessoa?
```

```
ref1 = Pessoa()
```

ref2 —————



```
var ref2: Pessoa?
```

```
ref2 = ref1
```

```
ref1 = nil
```

Automatic Reference Counting

strong

```
class Pessoa { }
```

ref1

```
var ref1: Pessoa?
```

```
ref1 = Pessoa()
```

ref2



```
var ref2: Pessoa?
```

```
ref2 = ref1
```

```
ref1 = nil
```

```
ref2 = nil
```

Automatic Reference Counting

strong

```
class Pessoa { }  
var ref1: Pessoa?  
ref1 = Pessoa()  
  
var ref2: Pessoa?  
ref2 = ref1  
  
ref1 = nil  
ref2 = nil
```

! ?

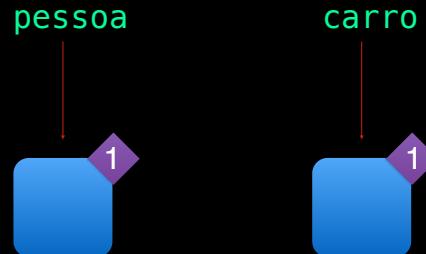
Ciclos de Referências

ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()
```

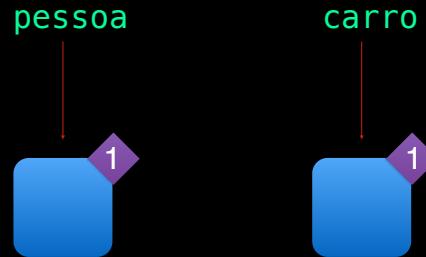


ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()
```

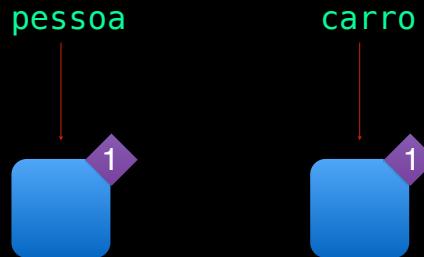


ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa
```

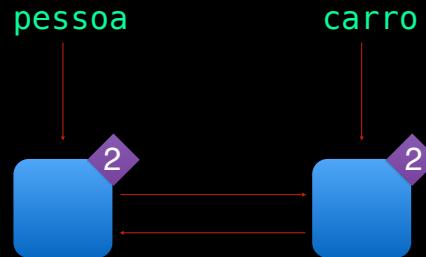


ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa
```

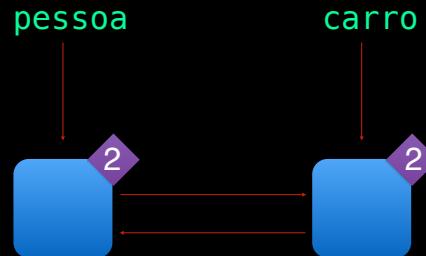


ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```

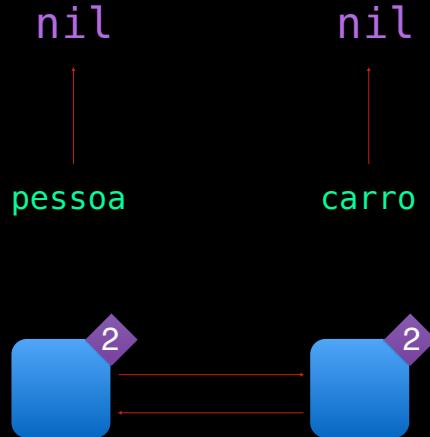


ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```



ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```

nil
|
pessoa
|
carro
nil



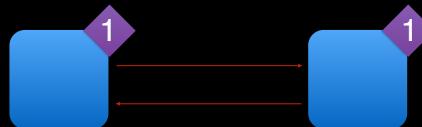
ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```

nil
|
pessoa
|
carro
nil





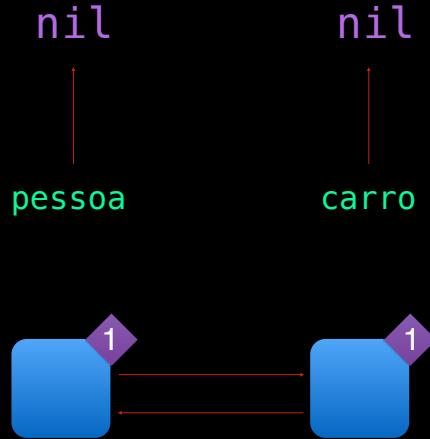
Memory Leak

ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```

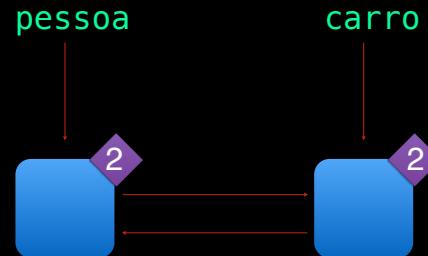


ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    weak var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```

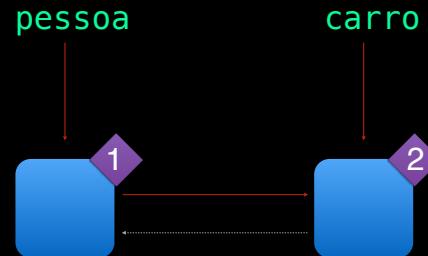


ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    weak var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```

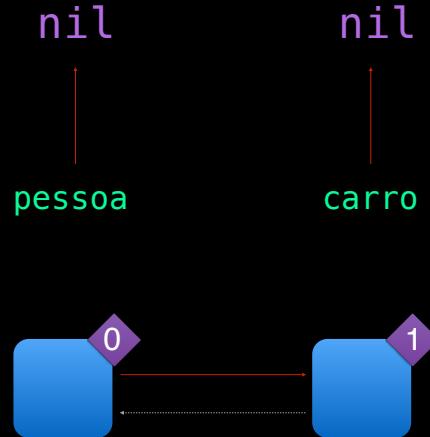


ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    weak var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```



ARC

(Vida Independente)

weak

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    weak var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa
```

nil

pessoa

nil

carro

nil



ARC

(Vida Independente)

weak

nil

nil

pessoa

carro

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    weak var proprietário:Pessoa?  
}
```

```
var pessoa: Pessoa? = Pessoa()
```

```
var carro: Carro? = Carro()
```

```
pessoa!.veículo = carro  
carro!.proprietário = pessoa
```

```
carro = nil  
pessoa = nil
```

ARC

(Vida Independente)

weak

nil

nil

pessoa

carro

```
class Pessoa {  
    var veículo: Carro?  
}  
  
class Carro {  
    weak var proprietário:Pessoa?  
}  
  
var pessoa: Pessoa? = Pessoa()  
var carro: Carro? = Carro()  
  
pessoa!.veículo = carro  
carro!.proprietário = pessoa  
  
carro = nil  
pessoa = nil
```



weak

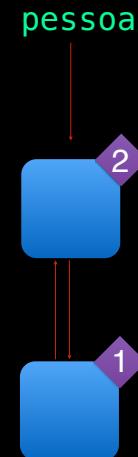
(Vidas Independentes)

ARC

(Vida Dependente)

unowned

```
class Pessoa {  
    var habilitação: Habilidade?  
}  
  
class Habilidade {  
    let titular: Pessoa  
  
    init(titular: Pessoa) {  
        self.titular = titular  
    }  
}  
  
var pessoa: Pessoa? = Pessoa()  
pessoa!.habilitação = Habilidade(titular: pessoa!)
```



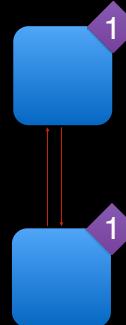
ARC

(Vida Dependente)

unowned

```
class Pessoa {  
    var habilitação: Habilidade?  
}  
  
class Habilidade {  
    let titular: Pessoa  
  
    init(titular: Pessoa) {  
        self.titular = titular  
    }  
}  
  
var pessoa: Pessoa? = Pessoa()  
pessoa!.habilitação = Habilidade(titular: pessoa!)  
  
pessoa = nil
```

nil ————— pessoa

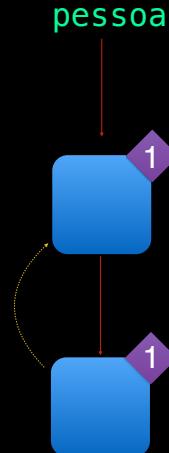


ARC

(Vida Dependente)

unowned

```
class Pessoa {  
    var habilitação: Habilidade?  
}  
  
class Habilidade {  
    unowned let titular: Pessoa  
  
    init(titular: Pessoa) {  
        self.titular = titular  
    }  
}  
  
var pessoa: Pessoa? = Pessoa()  
pessoa!.habilitação = Habilidade(titular: pessoa!)
```



ARC

(Vida Dependente)

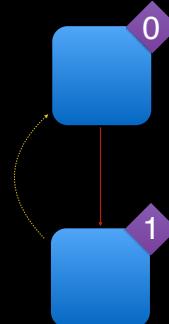
unowned

```
class Pessoa {  
    var habilitação: Habilidade?  
}
```

```
class Habilidade {  
    unowned let titular: Pessoa  
  
    init(titular: Pessoa) {  
        self.titular = titular  
    }  
}
```

```
var pessoa: Pessoa? = Pessoa()  
pessoa!.habilitação = Habilidade(titular: pessoa!)  
  
pessoa = nil
```

nil ————— pessoa



unowned

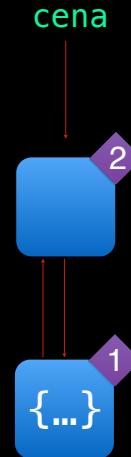
(Vida Dependente)

Closures

(Lista de Captura)

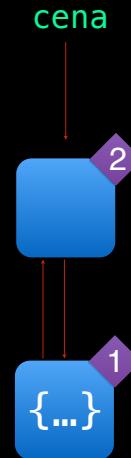
Closures

```
class Botão {  
    var action: (()->())?  
}  
  
class Cena {  
    let botão = Botão()  
    var contagem = 0  
  
    init() {  
        botão.action = {  
            contagem += 1  
        }  
    }  
}  
  
var cena:Cena? = Cena()
```



Closures

```
class Botão {  
    var action: (()->())?  
}  
  
class Cena {  
    let botão = Botão()  
    var contagem = 0  
  
    init() {  
        botão.action = {  
            self.contagem += 1  
        }  
    }  
}  
  
var cena:Cena? = Cena()
```



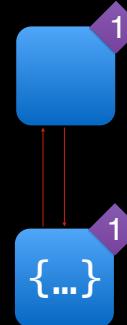
Closures

```
class Botão {  
    var action: (()->())?  
}
```

```
class Cena {  
    let botão = Botão()  
    var contagem = 0  
  
    init() {  
        botão.action = {  
            self.contagem += 1  
        }  
    }  
}
```

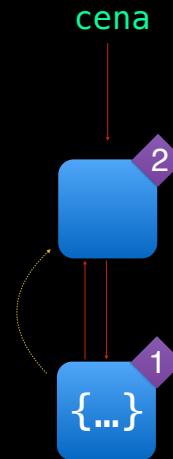
```
var cena:Cena? = Cena()  
  
cena = nil
```

nil —— cena



Closures

```
class Botão {  
    var action: (()->())?  
}  
  
class Cena {  
    let botão = Botão()  
    var contagem = 0  
  
    init() {  
        self.botão.action = { [weak self] in  
            self?.contagem += 1  
        }  
    }  
}  
  
var cena:Cena? = Cena()
```



Closures

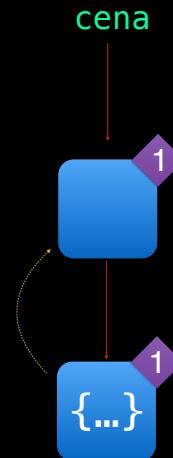
```
class Botão {  
    var action: (()->())?  
}
```

```
class Cena {  
    let botão = Botão()  
    var contagem = 0  
  
    init() {  
        self.botão.action = { [weak self] in  
            self?.contagem += 1  
        }  
    }  
}  
  
var cena:Cena? = Cena()
```



Closures

```
class Botão {  
    var action: (()->())?  
}  
  
class Cena {  
    let botão = Botão()  
    var contagem = 0  
  
    init() {  
        self.botão.action = { Lista de Captura [weak self] in  
            self?.contagem += 1  
        }  
    }  
}  
  
var cena:Cena? = Cena()
```



Closures

```
let closure: (()->())? = { [weak self] in
    if let strongSelf = self {
        strongSelf.view.alpha = 0.0
    }
}
```

Closures

```
let resource = UIViewController()  
  
let closure: (()->())? = { [weak self, weak resource] in  
    if let strongSelf = self {  
        strongSelf.view.alpha = 0.0  
    }  
  
    resource?.view.alpha = 0.0  
}
```

Prevenção

Cuidado 

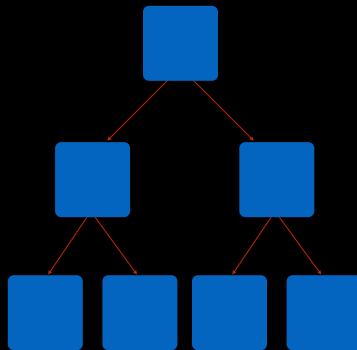
- Relações de Posse
- Referencias para classes
- Closures
- NSTimer
- UIAlertController

Use logs:

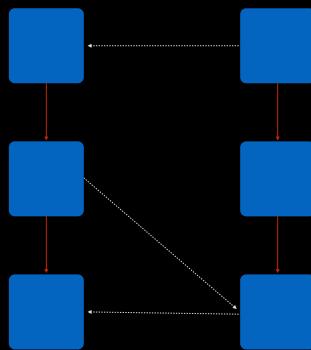
```
deinit {
    println("object is being deallocated")
}

-(void) dealloc
{
    NSLog(@"object is being deallocated");
}
```

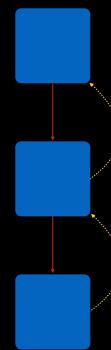
strong



weak



unowned





Value Types



Reference Types



Gerenciamento de Memória

Técnicas de DEBUG

Técnicas de DEBUG

- LLDB
- Breakpoints

**Quantas vezes você ja
precisou recompilar seu
código pra realizar um
simples teste?**

print(**valor**)

```
print(valor)
```

```
print("passou")
```



Tempo de Compilação em Swift

Alterações no Código !

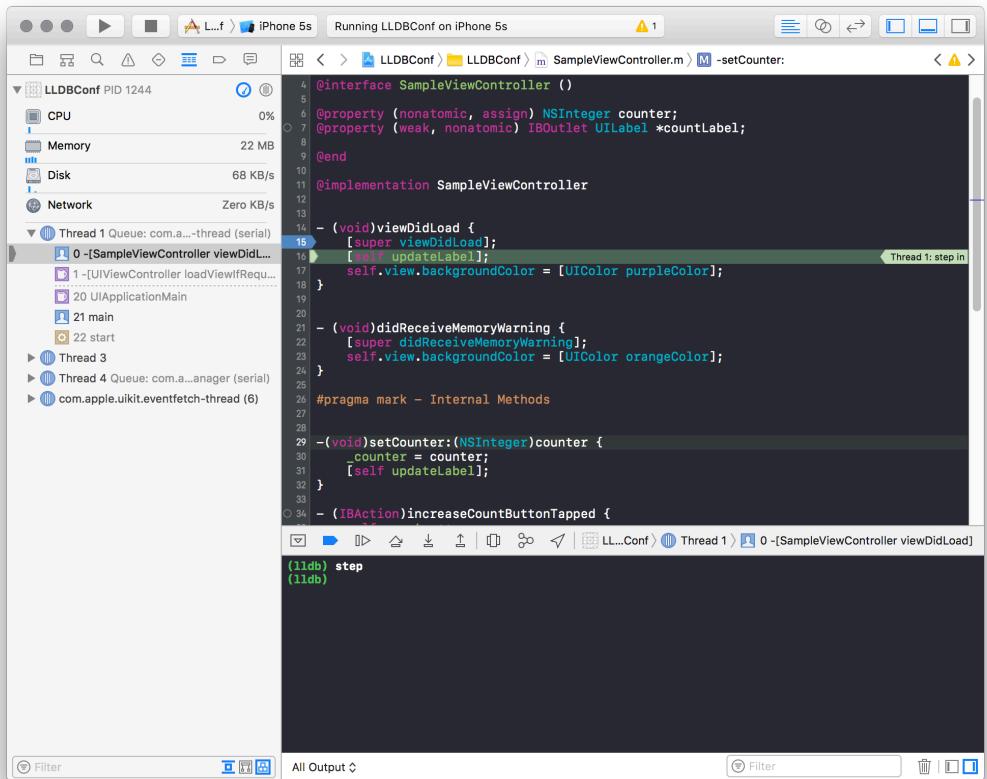
- Altamente problemática.
- E muito fácil esquecer uma mudança no código e ele acabar indo pra produção
- Até um simples `print()` em produção no lugar errado, pode causar problemas de performance e inclusive falhas de segurança.

Vamos usar o LLDB a nosso favor

LLDB

**LLDB é o debugger
default que vem no
Xcode.**

Breakpoints



The screenshot shows the Xcode interface during a debug session. The title bar indicates "Running LLDBConf on iPhone 5s". The left sidebar displays system monitoring for "LLDBConf PID 1244" with metrics for CPU, Memory, Disk, and Network. The main area shows the code for "SampleViewController.m". A red arrow points to line 15, which contains the code "[self.view.backgroundColor = [UIColor purpleColor];". This line is highlighted in green, indicating it is the current instruction being executed. The status bar at the bottom shows "(lldb) step".

```
interface SampleViewController ()  
{  
    @property (nonatomic, assign) NSInteger counter;  
    @property (weak, nonatomic) IBOutlet UILabel *countLabel;  
}  
  
implementation SampleViewController  
  
- (void)viewDidLoad {  
    [super viewDidLoad];  
    [self.updateLabel];  
    self.view.backgroundColor = [UIColor purpleColor];  
}  
  
- (void)didReceiveMemoryWarning {  
    [super didReceiveMemoryWarning];  
    self.view.backgroundColor = [UIColor orangeColor];  
}  
  
#pragma mark - Internal Methods  
  
-(void)setCounter:(NSInteger)counter {  
    _counter = counter;  
    [self updateLabel];  
}  
  
- (IBAction)increaseCountButtonTapped {
```

L...f iPhone 5s Running LLDBConf on iPhone 5s 1

LLDBConf PID 1244 CPU 0% Memory 22 MB Disk 68 KB/s Network Zero KB/s

Thread 1 Queue: com.apple.main-thread (serial)

- 0 -[SampleViewController viewDidLoad]
- 1 -[UIViewController loadViewIfRequired]
- 20 UIApplicationMain
- 21 main
- 22 start

Thread 3

Thread 4 Queue: com.apple.main-thread (serial)

com.apple.uikit.eventfetch-thread (6)

```
4 @interface SampleViewController ()  
5  
6 @property (nonatomic, assign) NSInteger counter;  
7 @property (weak, nonatomic) IBOutlet UILabel *countLabel;  
8  
9 @end  
10  
11 @implementation SampleViewController  
12  
13  
14 - (void)viewDidLoad {  
15     [super viewDidLoad];  
16     [self updateLabel];  
17     self.view.backgroundColor = [UIColor purpleColor];  
18 }  
19  
20  
21 - (void)didReceiveMemoryWarning {  
22     [super didReceiveMemoryWarning];  
23     self.view.backgroundColor = [UIColor orangeColor];  
24 }  
25  
26 #pragma mark - Internal Methods  
27  
28  
29 -(void)setCounter:(NSInteger)counter {  
30     _counter = counter;  
31     [self updateLabel];  
32 }  
33  
34 - (IBAction)increaseCountButtonTapped {  
35 }
```

(lldb) step
(lldb)

continue
step
stepi
finish

Print: O comando Print serve para printar uma variável, geralmente utilizado para printar tipos primitivos;

Print Object: Parecido com o print, mas geralmente é utilizado para printar objetos.

Expression: O expression vai basicamente rodar (evaluate) uma expressão que você digitar.

Watchpoint: Esse aqui é legal para acompanhar se um valor foi alterado. Digamos que você tem uma variável e quer ser avisado de toda vez que ela sofrer uma alteração.

p po e w

Print e Print Object

```
(lldb) p counter  
(NSInteger) $0 = 1
```

```
(lldb) p countLabel  
(UILabel *) $1 = 0x00007fefb650a7f0
```

```
(lldb) po self.countLabel  
<UILabel: 0x7fefb650a7f0; frame = (122 274; 76.5 20.5); text =  
'Count = 1'; opaque = NO; autoresizingMask = RM+BM;  
userInteractionEnabled = NO; layer = <_UILabelLayer:  
0x61800008f70>>
```

Expression

Expression

```
view.backgroundColor = .red
```

Expression

```
(lldb) e view.backgroundColor = .red  
(UICachedDeviceRGBColor *) $3 = 0x00006080000735c0
```

step

OU

```
e [CATransaction flush]
```

DEMO



Técnicas de DEBUG

Dicas de Segurança

Dicas de Segurança

- Armazenando Dados
- Networking
- Informações Sensíveis

Armazenando Dados

Não salve informações sensíveis do usuário no banco de dados, ou no `NSUserDefaults` ou em arquivos no bundle!!!!

Armazenando Dados

Salve informações sensíveis no Keychain!

```
// Salva  
KeychainWrapper.standard.set("_password_", forKey: "senhaUser")  
  
// Recupera  
KeychainWrapper.standard.string(forKey: "senhaUser")
```

Networking

Enable Application Transport Security
HTTPS protocol and TLS 1.2

SSL Pinning
Dificulta ataque "Man in the Middle"

Informações Sensíveis do App

```
static let url = "https://dev-api.oiwarren.com"  
static let apiKey = "key=yr03498trhg6r[d09832k3r865alkd"
```

Informações Sensíveis do App

```
static let url = "https://dev-api.oiwarren.com"  
static let apiKey = "key=yr03498trhg6r[d09832k3r865alkd"
```

→ SecurityExample.app strings SecurityExample

Informações Sensíveis do App

```
static let url = "https://dev-api.oiwarren.com"  
static let apiKey = "key=yr03498trhg6r[d09832k3r865alkd"
```

→ SecurityExample.app strings SecurityExample

T@"UILabel", N, W, VbudgetLabel

Data:

JSON:

<https://dev-api.oiwarren.com>

key=yr03498trhg6r[d09832k3r865alkd

_TtC15SecurityExample11AppDelegate

B32@0:8@16@24

window

T@"UIWindow", N, S, Vwindow

Informações Sensíveis do App

```
// Configuração fora do App
let o = Obfuscator(withSalt: [AppDelegate.self])
let bytes = o.bytesByObfuscatingString(string: apiKey)
/*[48, 54, 28, ..., 0, 72, 85, 86, 26, 52]*/

// Acesso no App
let apiKeyObfs: [UInt8] = [48, 54, 28, ..., 26, 52]
let original = o.reveal(key: apiKeyObfs)
/*key=yr03498trhg6r[d09832k3r865alkd*/
```

Extras

- Se seu app depende muito de segurança existem diversas técnicas, mas custam cara e exigem tempo.
- Debug e jailbreak detection pode ser uma opção.
- Obfuscação total de código.
- Não existe maneira fácil



Dicas de Segurança



warren

Txai Wieser

[linkedin.com/in/txaiwieser](https://www.linkedin.com/in/txaiwieser)

github.com/txaiwieser

twitter.com/txaiwieser

txai.wieser@icloud.com