

cit 

Hilton Pintor

Desenvolvedor (iOS/tvOS/watchOS)



hiltonpintor@gmail.com



// Aula 04

// Protocolos (II)

// extensões

Extensões **adicionam nova funcionalidade** a uma classe, estrutura, enumeração, ou protocolo existente.

É possível também extender tipos dos quais não temos acesso ao código fonte (***retroactive modeling***)

// extensões

- Adicionar **propriedades computadas**
- Definir **métodos**
- Prover novos **construtores**
- Definir **tipos aninhados**
- Conformar com **protocolo**

// extensões

Extensões podem adicionar novas funcionalidades a um tipo, mas **não podem sobrescrever** (*override*) funcionalidades existentes.

// nova classe

```
class Pessoa {  
    let nome: String  
    var anoNascimento: Int  
    var telefone: String?  
  
    init(nome: String, anoNascimento: Int, telefone: String? = nil) {  
        self.nome = nome  
        self.anoNascimento = anoNascimento  
        self.telefone = telefone  
    }  
}
```



```
// default value
```

```
var joao = Pessoa(nome: "João",  
                  anoNascimento: 1996)
```

```
var marina = Pessoa(nome: "Marina",  
                    anoNascimento: 1990,  
                    telefone: "94382712")
```

```
joao.telefone // nil  
marina.telefone // "94382712"
```

// estendendo classe

```
extension Pessoa {  
    var idade: Int {  
  
        let date = Date()  
        let calendar = Calendar.current  
        let year = calendar.component(.year, from: date)  
  
        return year - self.anoNascimento  
    }  
}
```

// acessando propriedade da extensão

joao.idade // 22

marina.idade // 28

```
// subclasse
```

```
class Professor: Pessoa {  
    var tituloP = "Prof."  
}
```

```
var jose = Professor(nome: "José", anoNascimento: 1980)  
jose.tituloP // "Prof."
```

// novo protocolo

```
protocol Mestre {  
    var tituloM: String { get }  
}
```

// estendendo protocolo

```
protocol Mestre {  
    var tituloM: String { get }  
}  
  
extension Mestre {  
    var tituloM: String {  
        return "Ms."  
    }  
}
```

```
// estendendo classe a para aderir a protocolo
```

```
extension Professor: Mestre { }
```

```
jose.tituloM // "Ms."
```

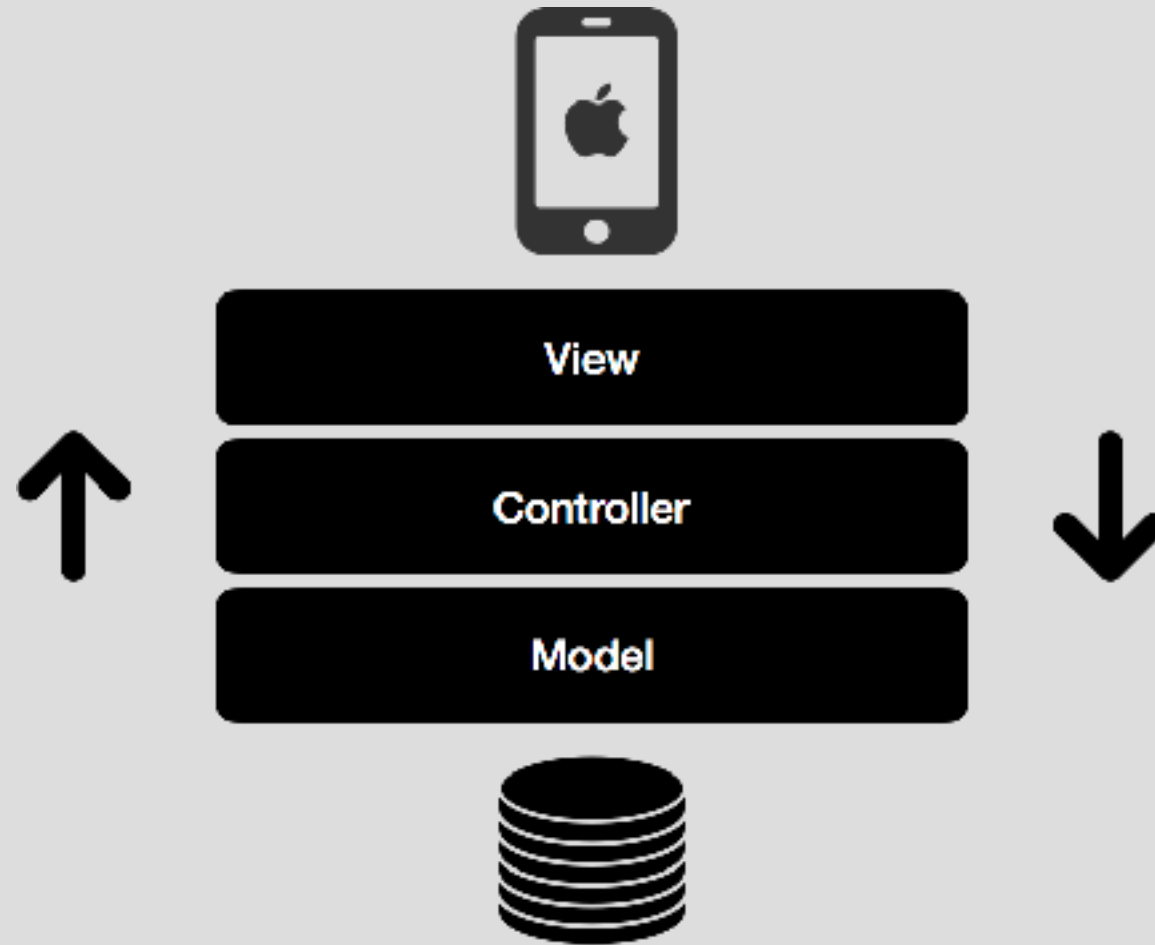
// MVC

// MVC

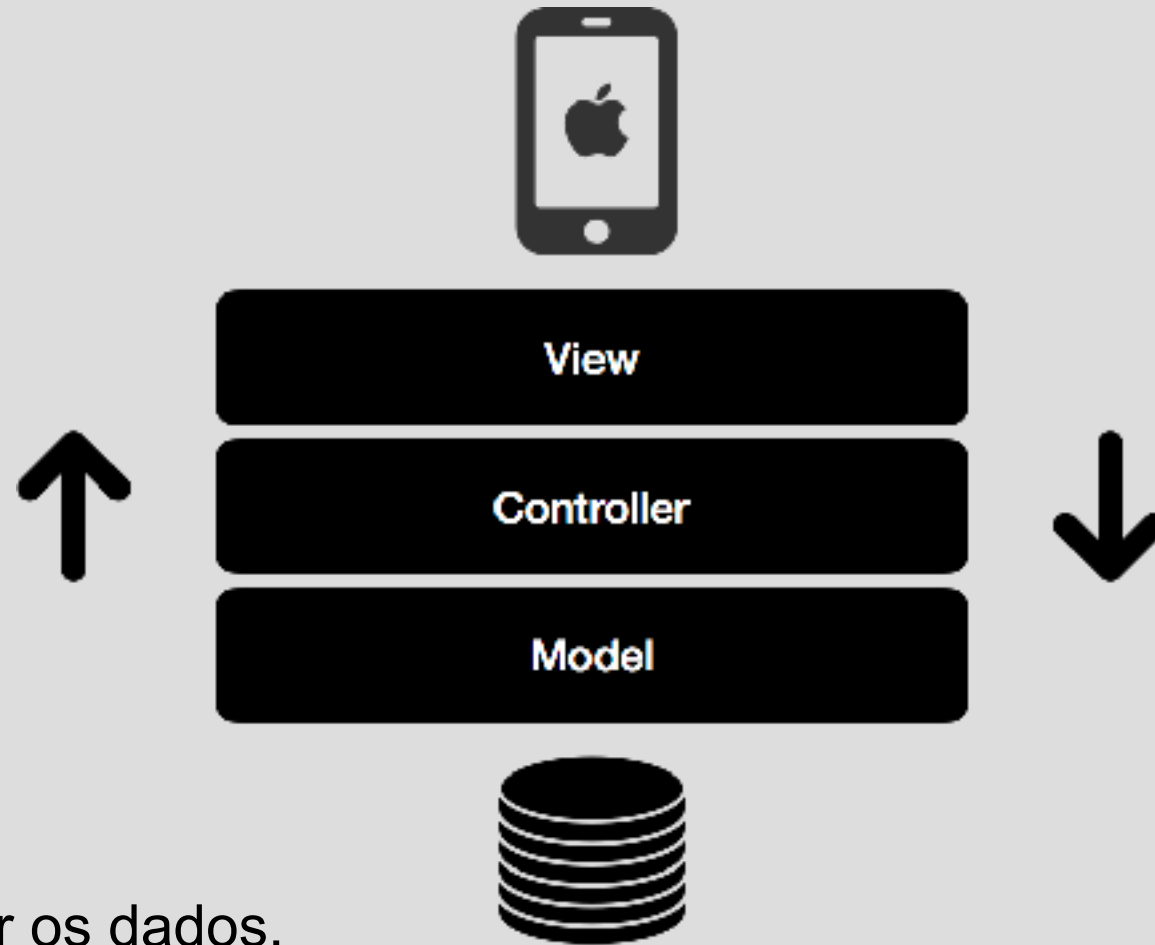
Para desenvolver um aplicativo, a arquitetura mais comumente utilizada, e recomendada pela própria Apple é o padrão ***Model-View-Controller*** (MVC).

Nessa arquitetura temos **três camadas** principais com responsabilidades definidas para nos ajudar a ter um código organizado e menos propenso a erros e inconsistências.

// MVC

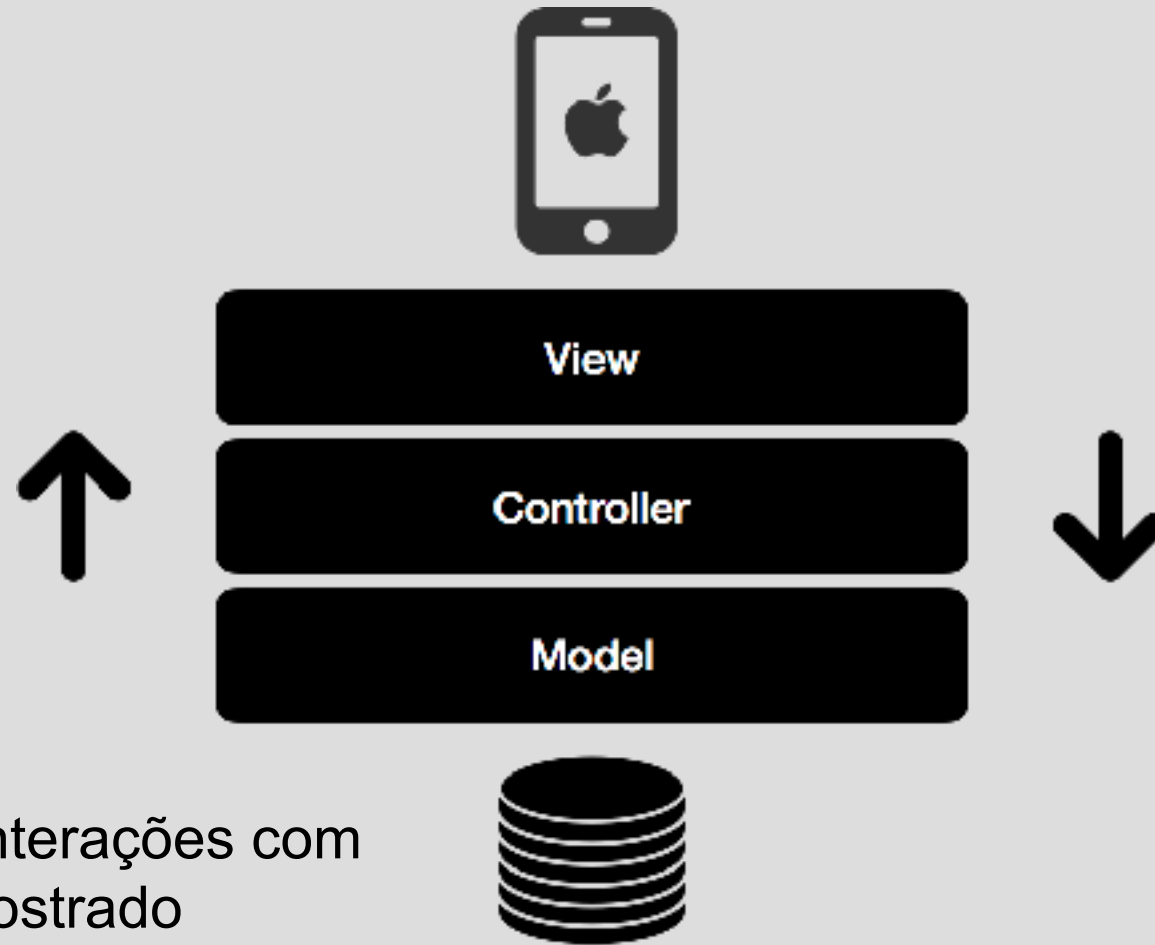


// MVC



Modelo (*model*),
responsável por manter os dados,
objetos, comunicação com a rede,
e regras de negócio da aplicação.

// MVC



Visão (*view*), responsável por todas interações com o usuário, tudo que é mostrado graficamente no iPhone. É onde os dados do modelo são mostrados e possivelmente editados.

// MVC



Controlador (*controller*),
Responsável por interpretar as ações
feitas pelo usuário nas views,
propagando mudanças ou criação de
dados para o modelo. Também é
responsável por atualizar as views de
acordo com os dados do modelo.

// UIKit

// UIKit

Framework para construir e gerenciar
interfaces gráficas orientadas a **eventos**,
em suas aplicações iOS.



// Views e Controls

Apresentam o seu **conteúdo** na tela, em maneiras específicas e definem **interações** permitidas com este conteúdo.



// UIView



View - Represents a rectangular region in which it draws and receives events.

//



View - Represents a rectangular region in which it draws and receives events.

Desenho e Animação

- Views desenharam conteúdo em sua área retangular usando UIKit ou Core Graphics.
- Algumas propriedades de uma view podem ser animadas.

//



View - Represents a rectangular region in which it draws and receives events.

Layout e subviews

- Views podem ter várias subviews.
- Views podem ajustar posição e tamanho de suas subviews.
- Podem utilizar Auto-layout para fazer o posicionamento responsivo



//



View - Represents a rectangular region in which it draws and receives events.

Event handling

- Views são subclasses de UIResponder podendo responder a toques e outros eventos.
- Views podem adicionar reconhecedores de gestos para lidar com gestos comuns.

// UILabel

Label **Label** - A variably sized amount of static text.

//

Label **Label** - A variably sized amount of static text.

```
class UILabel : UIView {  
    var text: String? // default is nil  
    // ...  
}
```

// UIButton

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

//

Button

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

```
class UIButton : UIControl{

    func setTitle(_ title: String?,
                  for state: UIControlState)
        // default is nil. title is assumed to be single line

        // ...

}
```

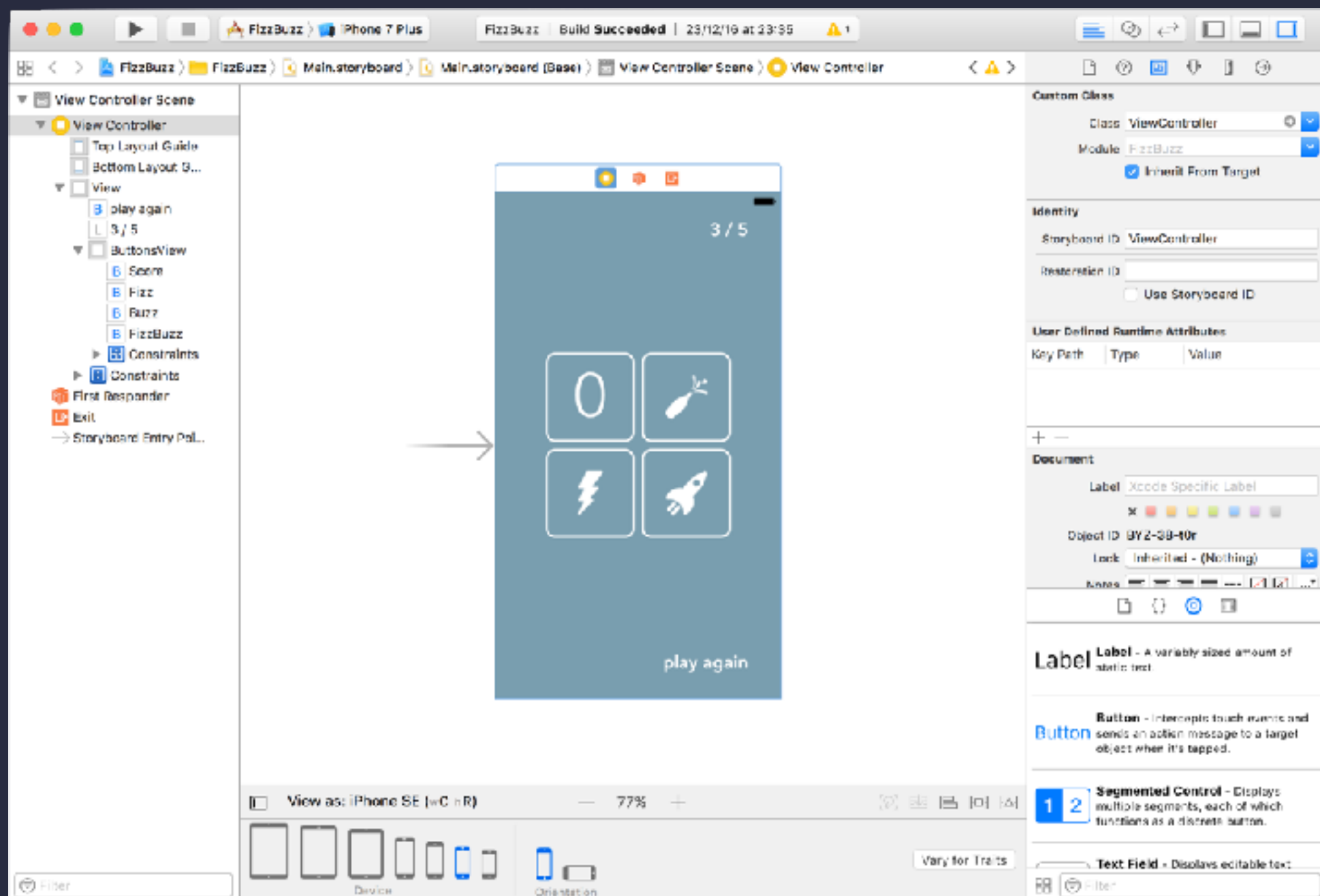
// StoryBoard

// storyBoard

Interface gráfica para montar suas GUIs

// storyboard

Uma forma mais fácil de dispor as views e suas subviews, customizá-las, e ver o resultado enquanto faz.



// ViewController

// viewControllers

View controllers são os fundamentais para a estrutura interna de um app.

Todo app tem pelo menos um view controller, ou vários.

Cada viewController gerencia uma porção da GUI, além das interações entre a interface e os dados.

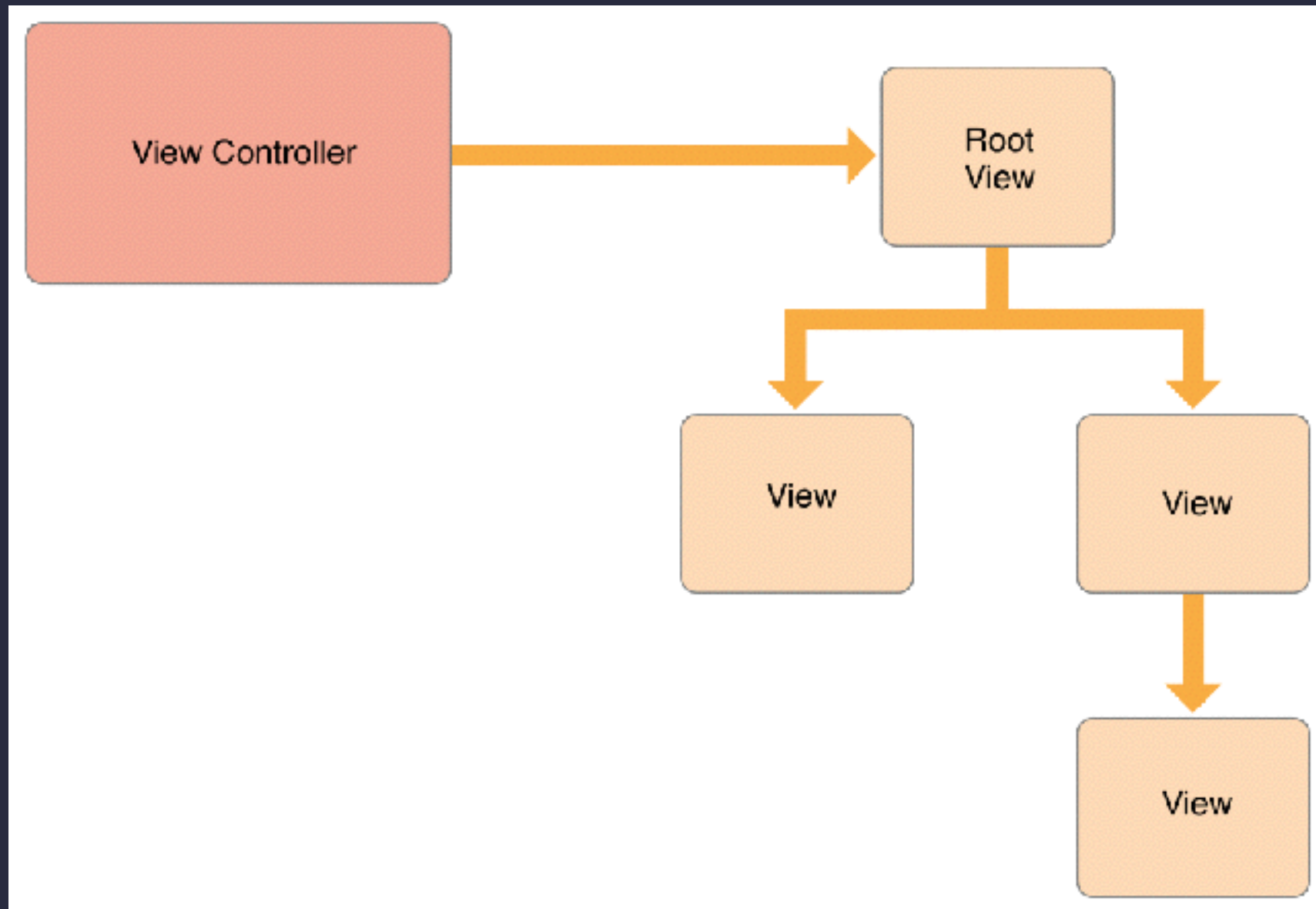


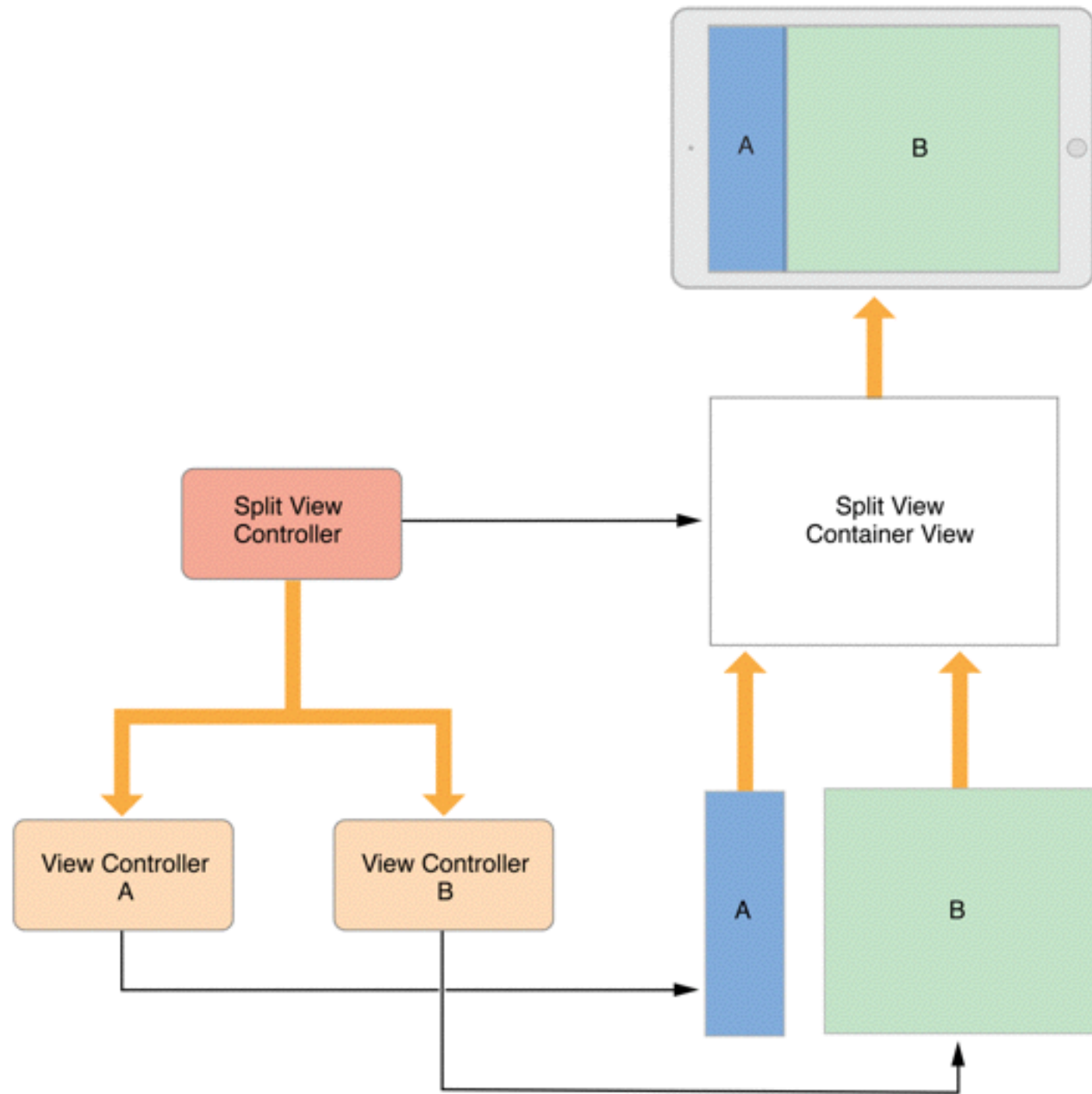
// viewControllers

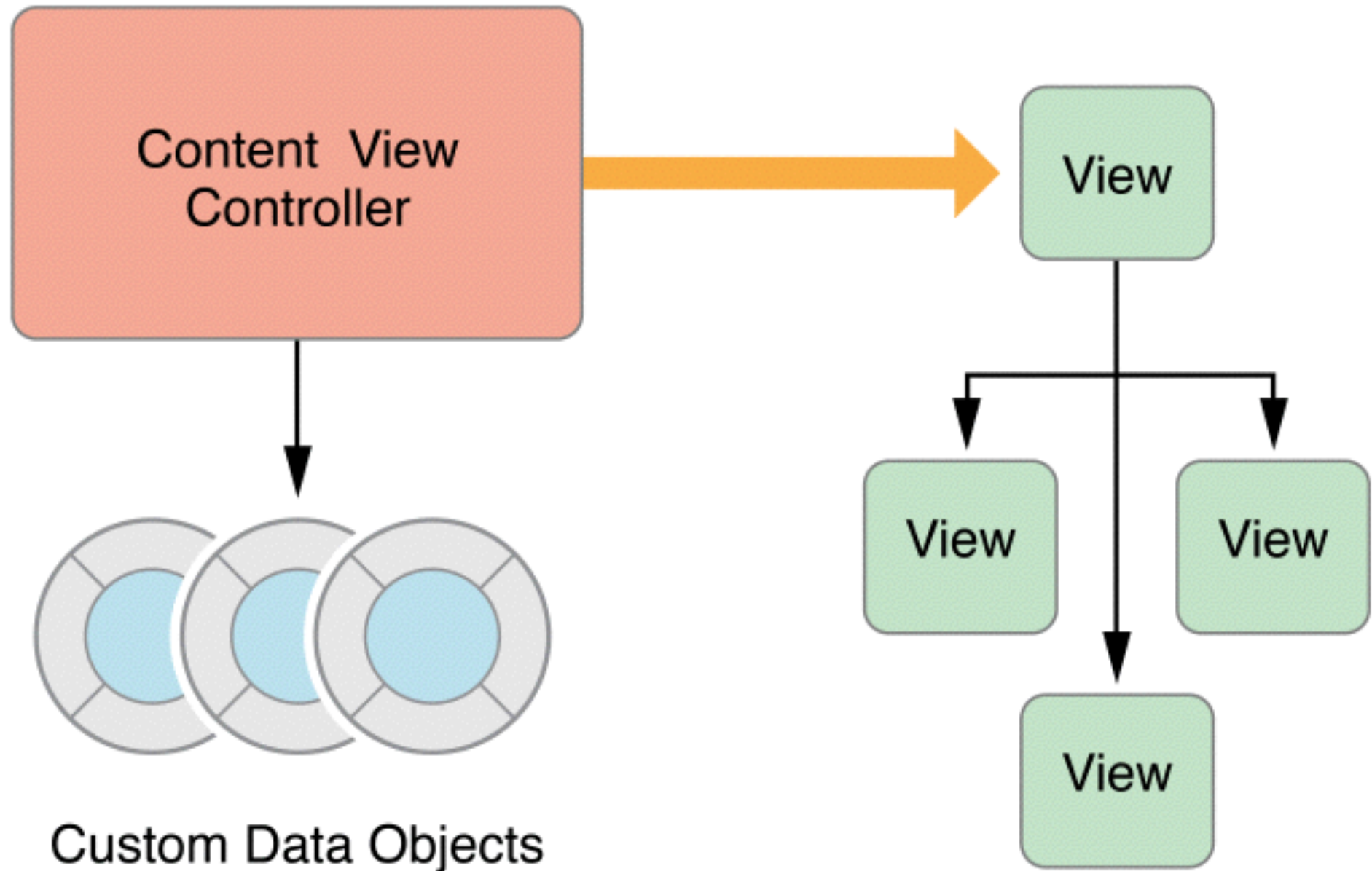
A classe **UIViewController** define métodos e propriedades para gerenciar as views, tratar eventos, transicionar entre view controllers, e coordenar com outras partes do app.

Fazemos subclasses de **UIViewController** para implementar o comportamento customizado da nossa aplicação.



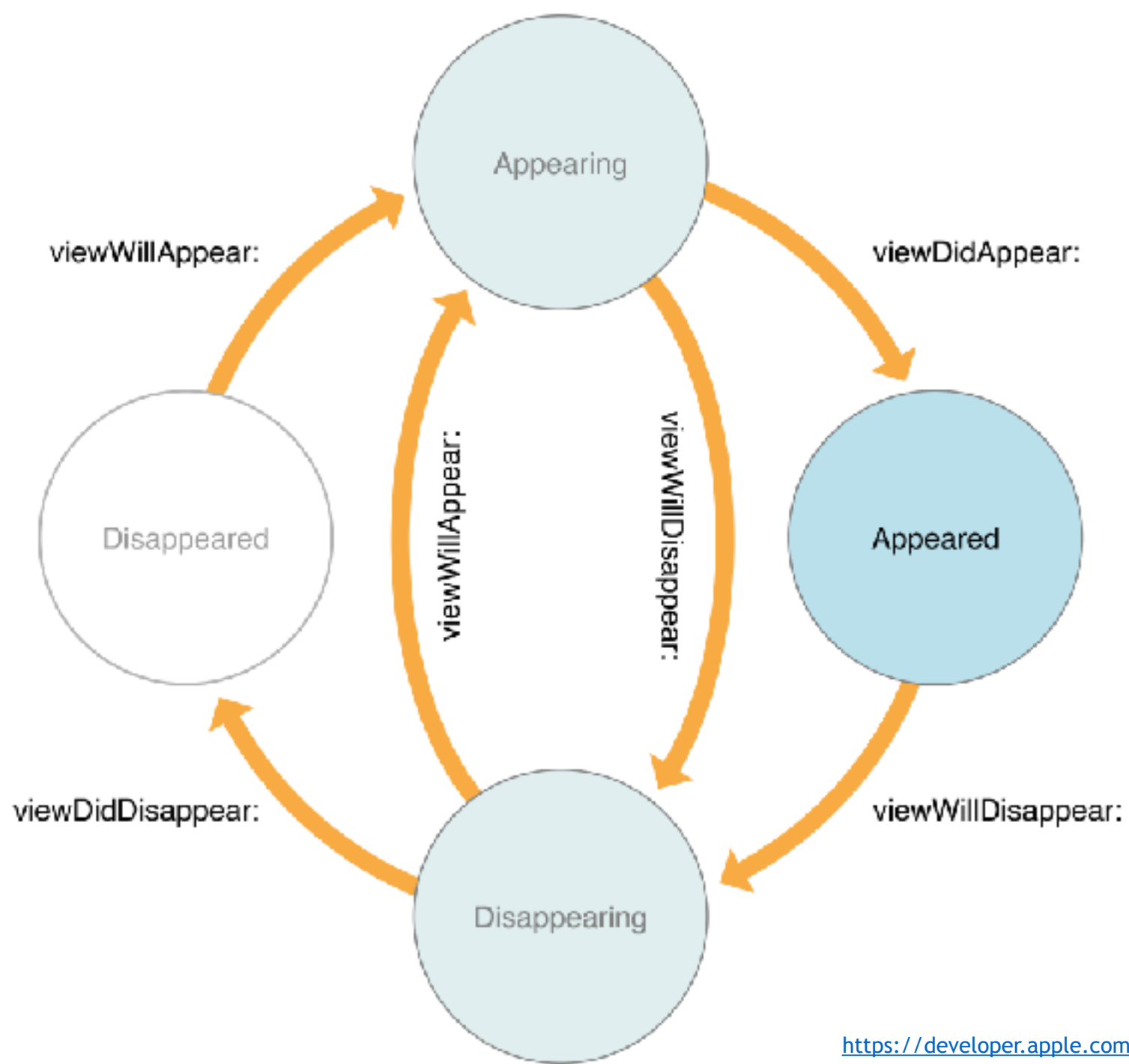






// viewControllers

View controllers tem um ciclo de vida, com métodos que são chamados sempre que algum marco acontece.



// viewControllers

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
        typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

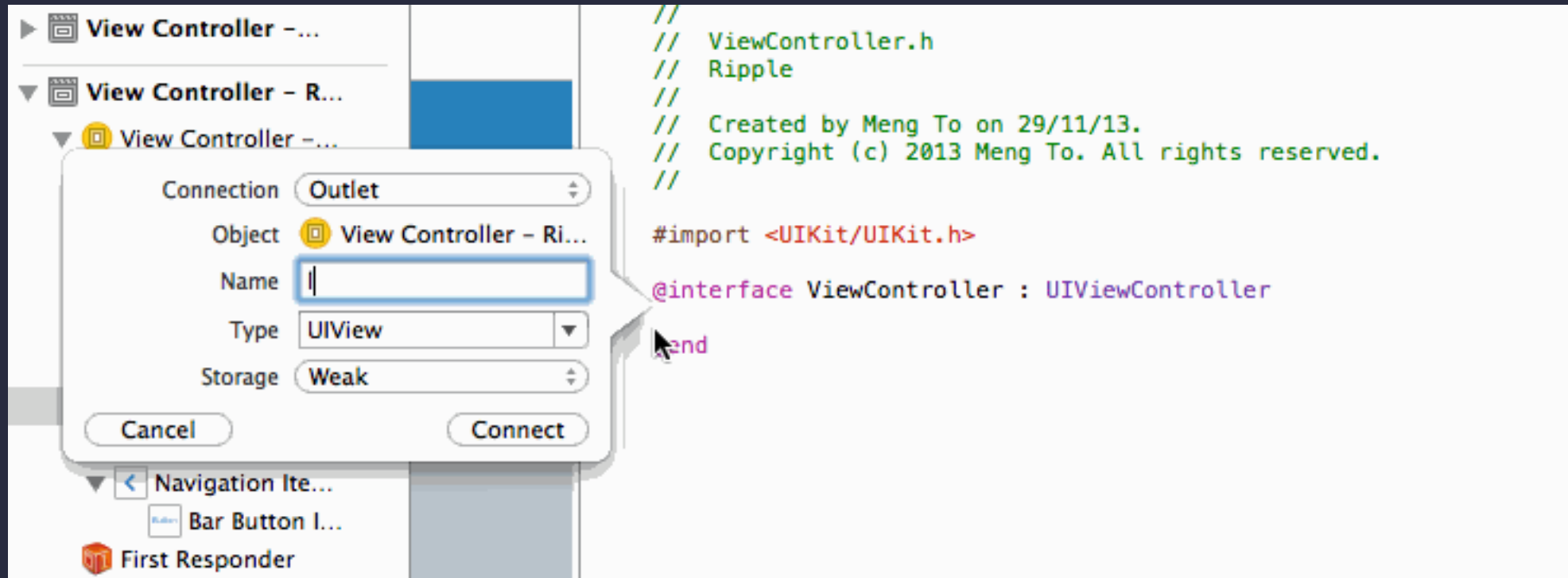
// ViewController/Storyboard

// problemática

Como dizer que um ViewController é responsável por uma view?

Como referenciar views adicionadas ao StoryBoard pelo código (ex: mudar conteúdo de um label, ação ao apertar um botão, ...)?

// IBOutlets & IBActions



// IBOutlet

Dá um nome a uma view do StoryBoard para que possa ser acessada por código no ViewController

// IBAction

Configura funções a serem executadas quando acontece alguma ação.

// Exercício

// Exercício 09

Button and Label

1. No StoryBoard:

1. adicione um botão
2. adicione uma label

2. No view controller:

- 2.1 ao **abrir o app**, mude o texto da label para a **data atual**
- 2.2 ao **apertar o botão**, mude o **texto da label** para a quantidade de vezes que o botão já foi apertado

// Delegate

// Delegate

Delegação é um padrão de projeto que permite que uma classe ou estrutura, passe (**delegue**) parte de suas **responsabilidades** para uma instância de um outro tipo

// Delegate

1. Definição do protocolo
2. Implementar protocolo em uma classe
3. adicionar propriedade delegate
4. chamar métodos do delegate

// Delegate

```
protocol DiceGame {  
    var dice: Dice { get }  
    func play()  
}
```

// Delegate

```
protocol DiceGameDelegate {  
    func gameDidStart(_ game: DiceGame)  
    func game(_ game: DiceGame,  
didStartNewTurnWithDiceRoll diceRoll: Int)  
    func gameDidEnd(_ game: DiceGame)  
}
```

// Delegate

```
class SnakesAndLadders: DiceGame {  
    // ...  
    var delegate: DiceGameDelegate?  
}
```

// Delegate

```
class DiceGameTracker: DiceGameDelegate {
    var numberOfTurns = 0
    func gameDidStart(_ game: DiceGame) {
        numberOfTurns = 0
        if game is SnakesAndLadders {
            print("Started a new game of Snakes and Ladders")
        }
        print("The game is using a \(game.dice.sides)-sided dice")
        "\(game.dice.sides)-sided dice"
    }

    func game(_ game: DiceGame, didStartNewTurnWithDiceRoll diceRoll: Int) {
        numberOfTurns += 1
        print("Rolled a \(diceRoll)")
    }

    func gameDidEnd(_ game: DiceGame) {
        print("The game lasted for \(numberOfTurns) turns")
    }
}
```

// Delegate

```
let tracker = DiceGameTracker()  
let game = SnakesAndLadders()  
game.delegate = tracker  
game.play()
```

// AppDelegate

// Exercício

// Exercício 10

Text Field

Faça um App com um botão e um campo de texto.

Inicie o **botão** como **desabilitado**.

Quando tiver texto escrito no campo de texto, **habilite** o **botão**.

Ao **apertar** o **botão**:

- se o usuário tiver digitado uma cor válida, modifique a **cor** do **background** da **view**.
- caso contrário, **notifique** o usuário que escreveu uma **cor inválida**

UIColor e shouldChangeCharactersIn range 

DÚVIDAS

