



G's ACADEMY

Dev16 iOS day2

2020.5.23

Naoki Kameyama

わほーい



iOSアプリ開発は、

1

直感的！

2

応用が効く！

3

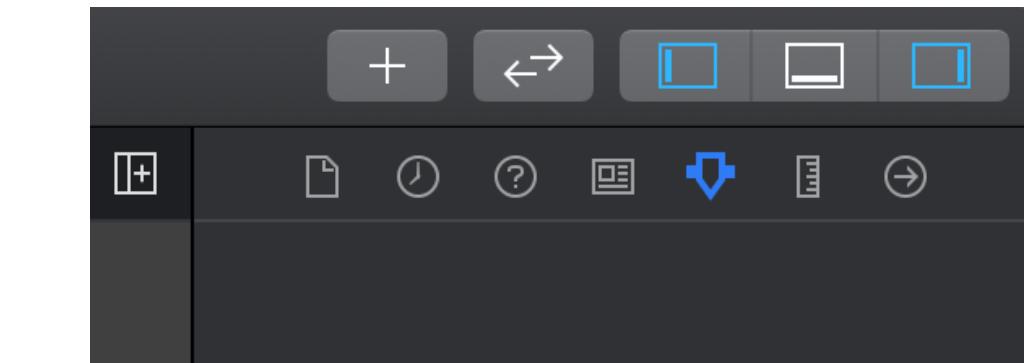
楽しい！

Day1の復習(1/2)

UIパーツをStoryboardに配置する方法

1

- Storyboardを開いてXcode右上の[+]をクリック
- はじめから多くのUIパーツが用意されている
- Autolayoutで相対的に位置を指定してゆく



2

ViewControllerと接続する

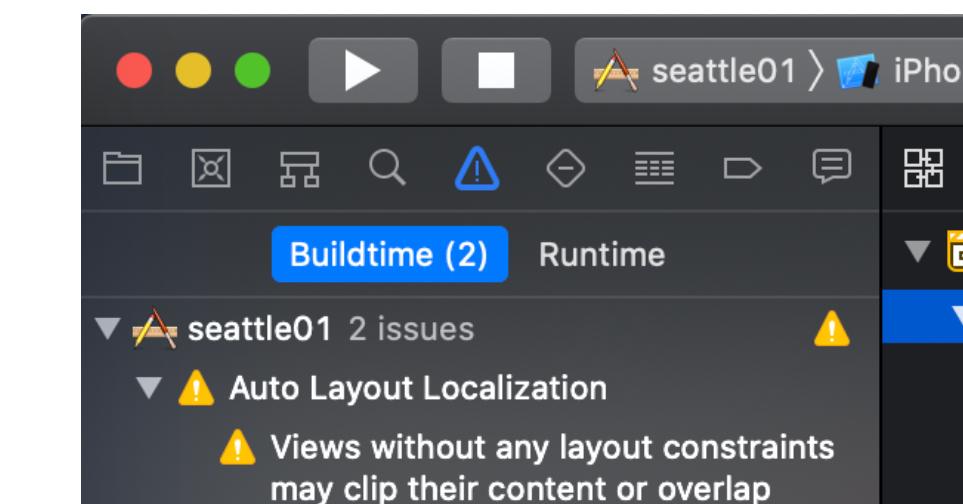
Point!

- [option]を押しながらファイルを選択すると、左右に画面が開く
- [control]を押しながらUIパーツをViewControllerへdrag&drop

3

シミュレーターを起動する

- Xcode左上の再生ボタンからシミュレーターを起動
- [command + R]でショートカットすると楽



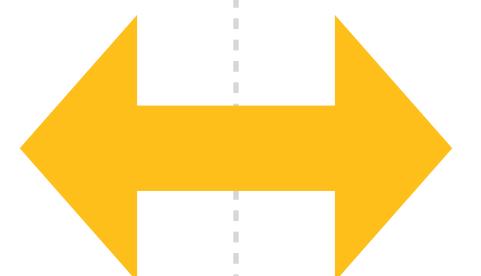
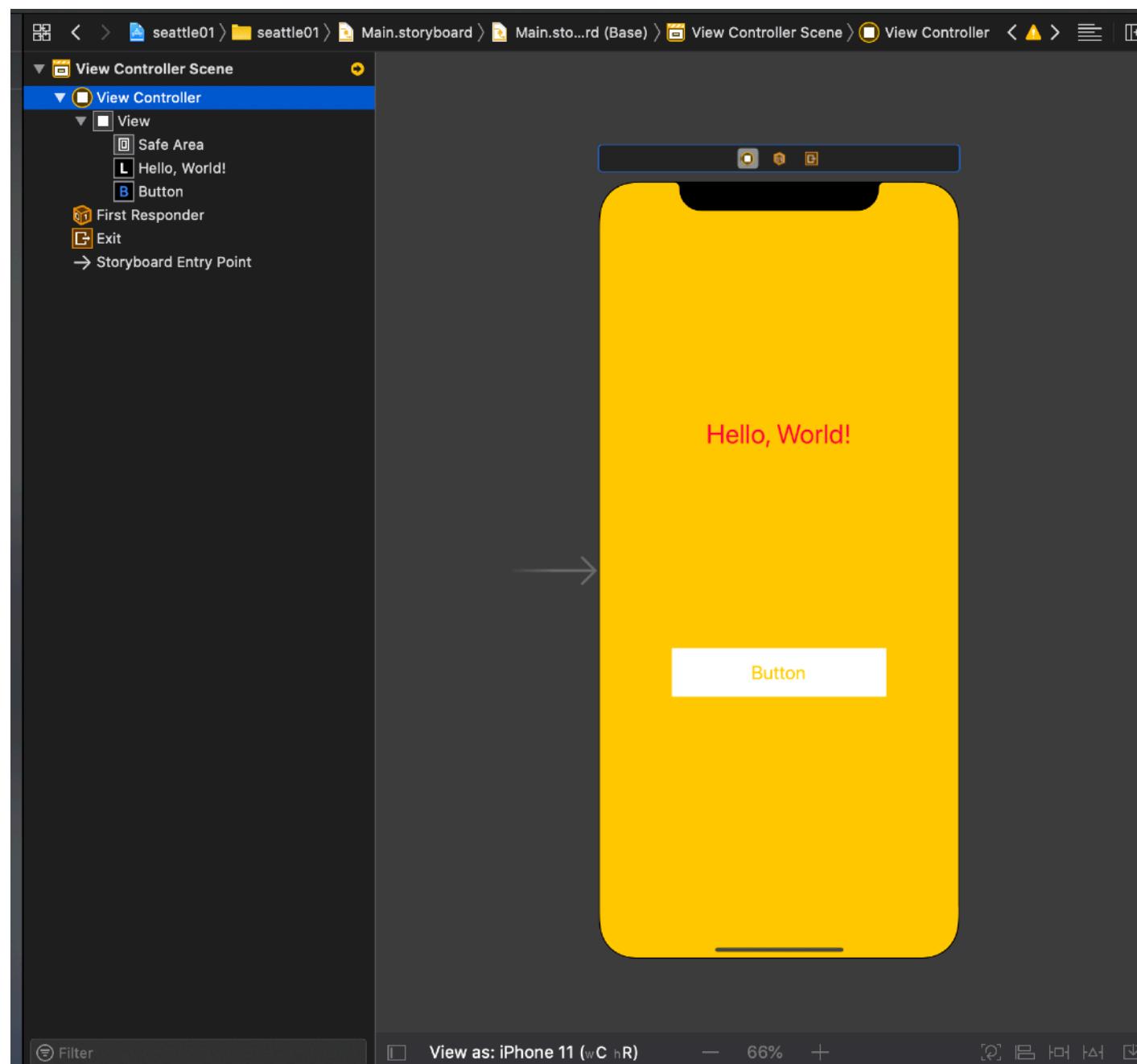
StoryboardとViewController

Storyboard

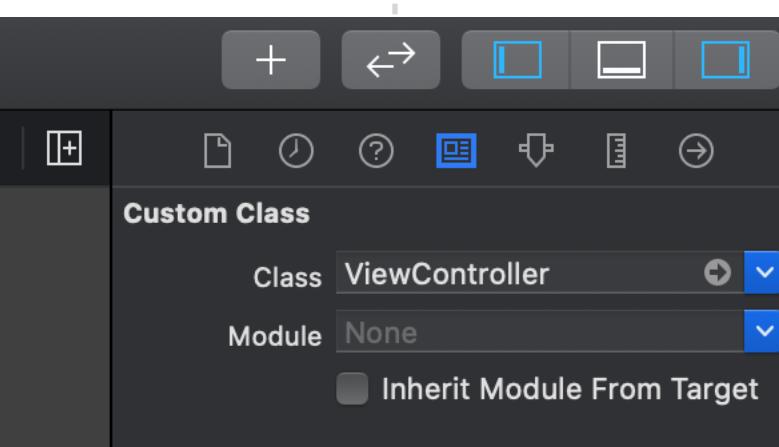
- PowerpointのようにUIを配置したり、位置を調整
 - よく利用する設定値(テキスト色など)を設定できる
 - = この一つ一つの設定値のことを"プロパティ"と言う
- Ex) UILabelのfontColorのプロパティを赤色に変える

ViewController

- 画面の表示を制御(いじくる)役割を持つファイル
 - Ex) 画面を表示する直前にテキストの中身を変えるボタンがタップされたときに○○を行う
 - 画面が消える直前に画像を削除する



関連付ける
必要あり



```
8
9 import UIKit
10
11 class ViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15         // Do any additional setup after loading the view.
16     }
17
18 }
19 }
```

Day1の復習(2/2)

変数と定数 型定義

1

- varとlet
- String, Int, Bool, Double
- 型推論で型を省略可

四則演算

2

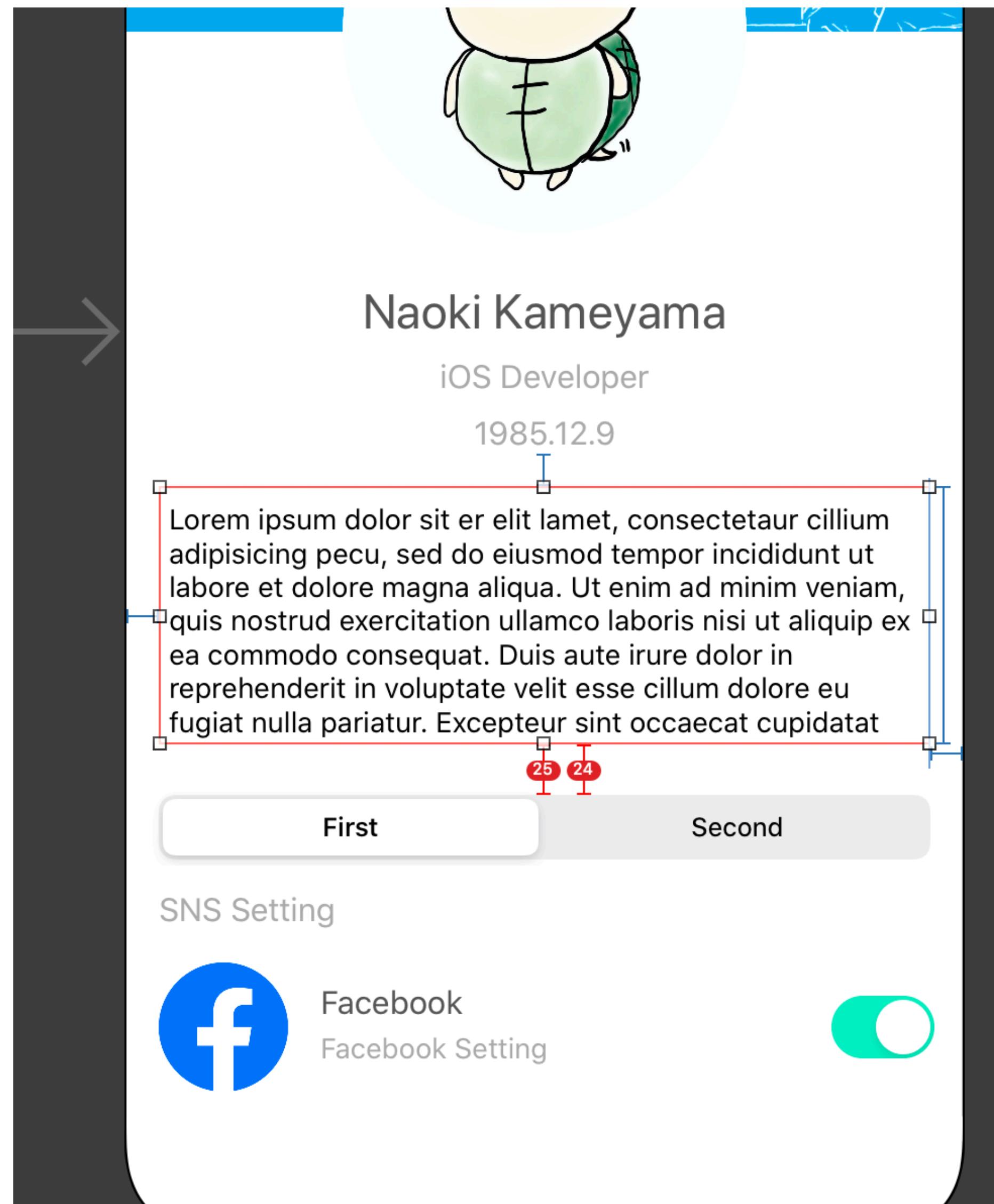
- +, -, *, /, %
- 型が異なるもの同士の計算はできない

配列、辞書、関数

3

- [String], [Int: String]
- func 関数名(引数名: 型) {処理}
- func 関数名(引数名: 型) -> 戻り値型 {処理}

Autolayoutエラー地獄



まずは落ち着きましょう！

Constraintが多くとも、少なくとも
赤いエラーとなります。

位置を固定するためには、
基本的には**4つのConstraint**が必要です。

* 固定されている箇所から

* **UILabel**など大きさが勝手に決まるものは
2つで良いものもあります。

* 自動補完は信じないで！

今日やること

1

プログラミング言語 Swiftの"基本"を理解する

- Optional型
- (後半)Classとインスタンス

2

画面遷移の作り方を理解する

- storyboardで遷移する vs コードで遷移する

3

カウントアップアプリをつくってみよう

- これまでの知識でアプリをつくろう
- データの保存 UserDefaults

Swiftについて学ぶ

Optionalについて知ろう(1/2)

オプショナル型とは**変数にnil(何もない)の代入を許す型**のこと

非オプショナル型

```
var firstName: String = nil // エラー
```

String

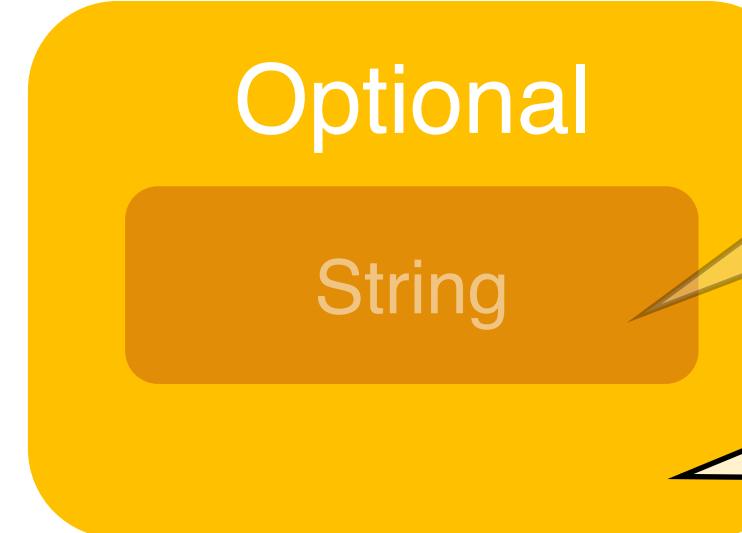
絶対nilでは
ないよ

オプショナル型 ...【?】をつけて宣言*

```
var firstName: String? = nil // OK
```

値がオプショナルで
包み込まれた状態をイメージする

*`var hoge: String!`と宣言する有値オプショナルというものもある



絶対nilでは
ないよ

nilが
入(って)るかも
しれないよ

非オプショナル型にオプショナル型やnilを代入することはできない

Optionalについて知ろう(2/2)

失敗する可能があるときは値がオプショナル型になる

例えば....

```
let tokyoOlympic = "2020"

//必ずIntに変換できるかわからないので、yearやcityはIntのOptionalになる
let year = Int(tokyoOlympic) //Optional(2020)
let city = Int("Tokyo") //nil
```

オプショナルを普通の値として使いたい場合

オプショナルのラップから**中身を取り出してあげる必要**がある

中身がある場合

-> 中身の値が取れる

中身がない(nilの)場合

-> nilが取れる

どうやるかは次ページ

Optionalを取り出す方法

- ①強制アンラップ(値が絶対あるときしかダメ)

```
print(getPoint!)
```

- ②if letによるアンラップ

```
if let _getPoint = getPoint {  
    print(_getPoint) //nilでない場合だけここにくる  
}
```

- ③guardによるアンラップ(多分一番使う) Point!

```
guard let _getPoint = getPoint else { return }  
print(_getPoint) //nilでない場合だけここにくる
```

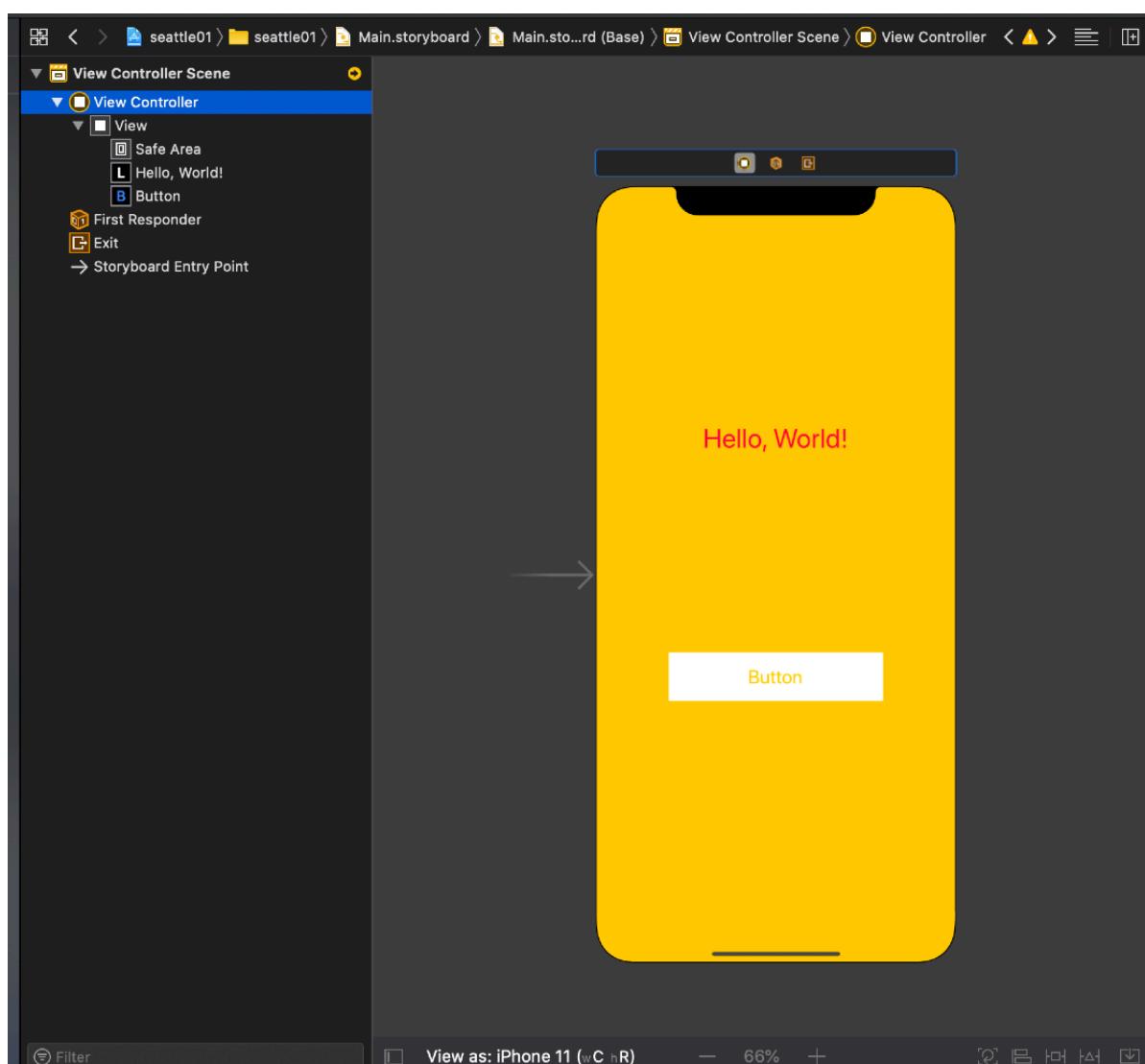
今はこんななんがあるよ～程度で
次回以降の授業で実際に使っていきながら覚えていきましょう

画面遷移をやってみよう

アプリ開発

Storyboard

- 画面にUIパーツを配置
- 相対的に位置を指定



ViewController

- UIパーツの設定値を指定
(Ex: 文字色を赤にする)
- Action後の動作を指定
(Ex: タップされたときの動作)
- 指定するときの言語がSwift

```
8 import UIKit
9
10 class ViewController: UIViewController {
11
12     override func viewDidLoad() {
13         super.viewDidLoad()
14         // Do any additional setup after loading the view.
15     }
16
17
18 }
```

画面遷移

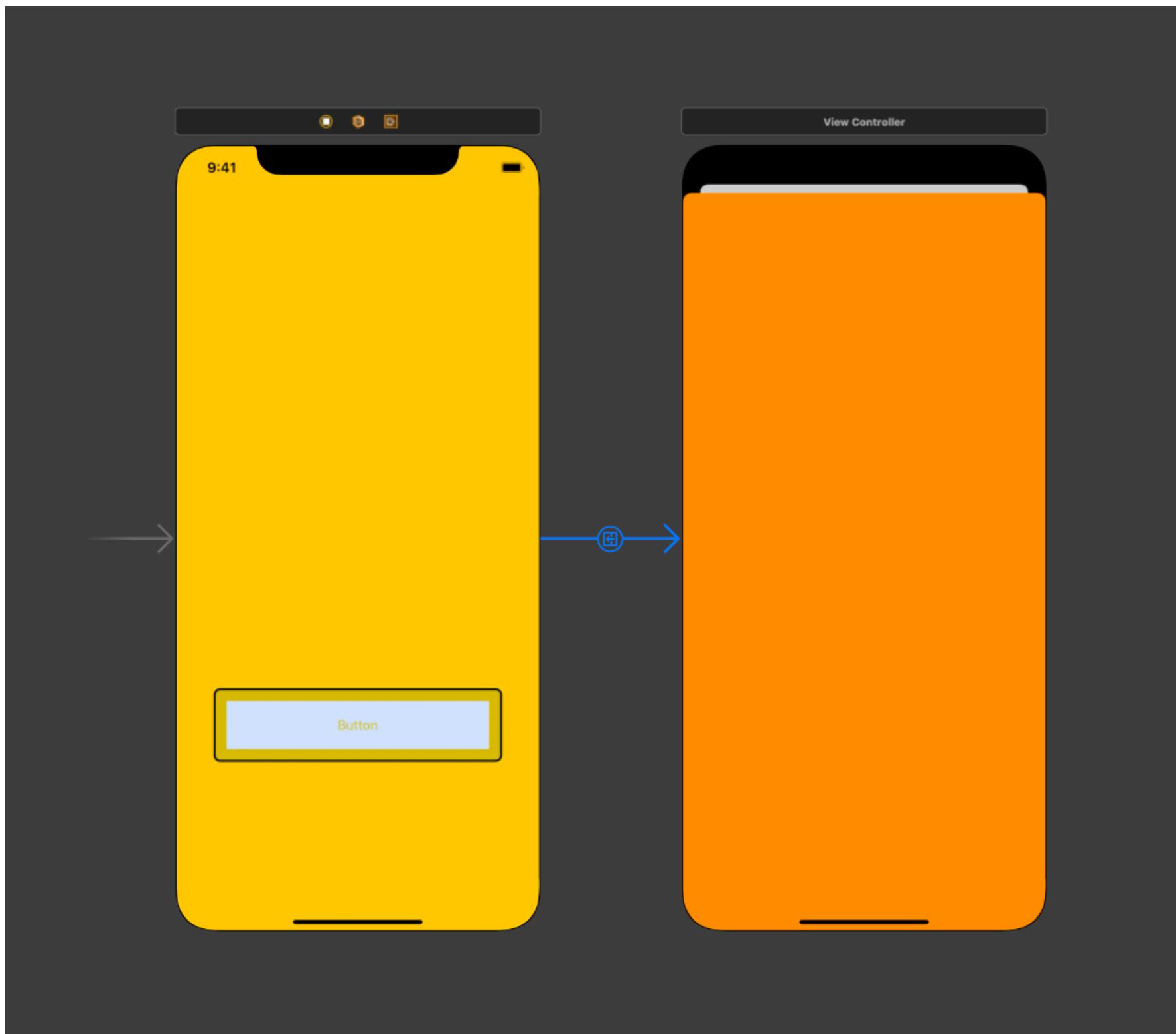
- Push: 横に遷移
- Modal: 下からニヨキッと



画面遷移する方法 2種類

Storyboardで画面遷移

- [control]押しながら遷移先のViewControllerを選択
- 直感的で分かりやすい一方で、画面数が増えると、スパゲッティStoryboardが生まれてしまう。

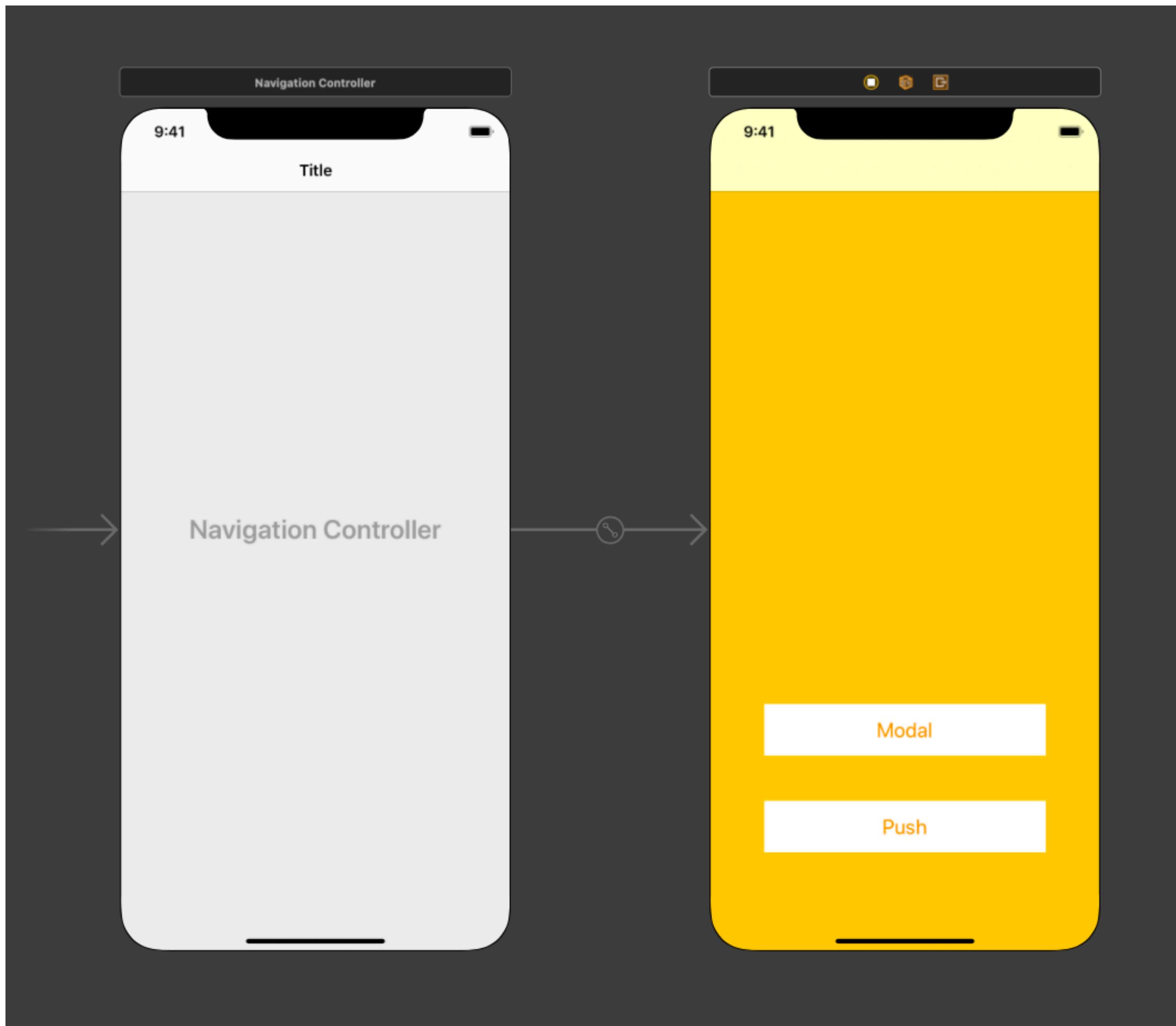


Codeで画面遷移

- 画面遷移でコードが必要なので、一見難しそう。
- やってみるとパターンが少なく楽。2個前の画面に戻るなども柔軟に対応できる。

```
17
18 @IBAction func tappedModalButton(_ sender: Any) {
19     let vc = SecondViewController()
20     self.present(vc, animated: true, completion: nil)
21 }
22
23 @IBAction func tappedPushButton(_ sender: Any) {
24     let vc = ThirdViewController()
25     navigationController?.pushViewController(vc, animated: true)
26 }
27 }
28
29
```

Codeで画面遷移(1/4)



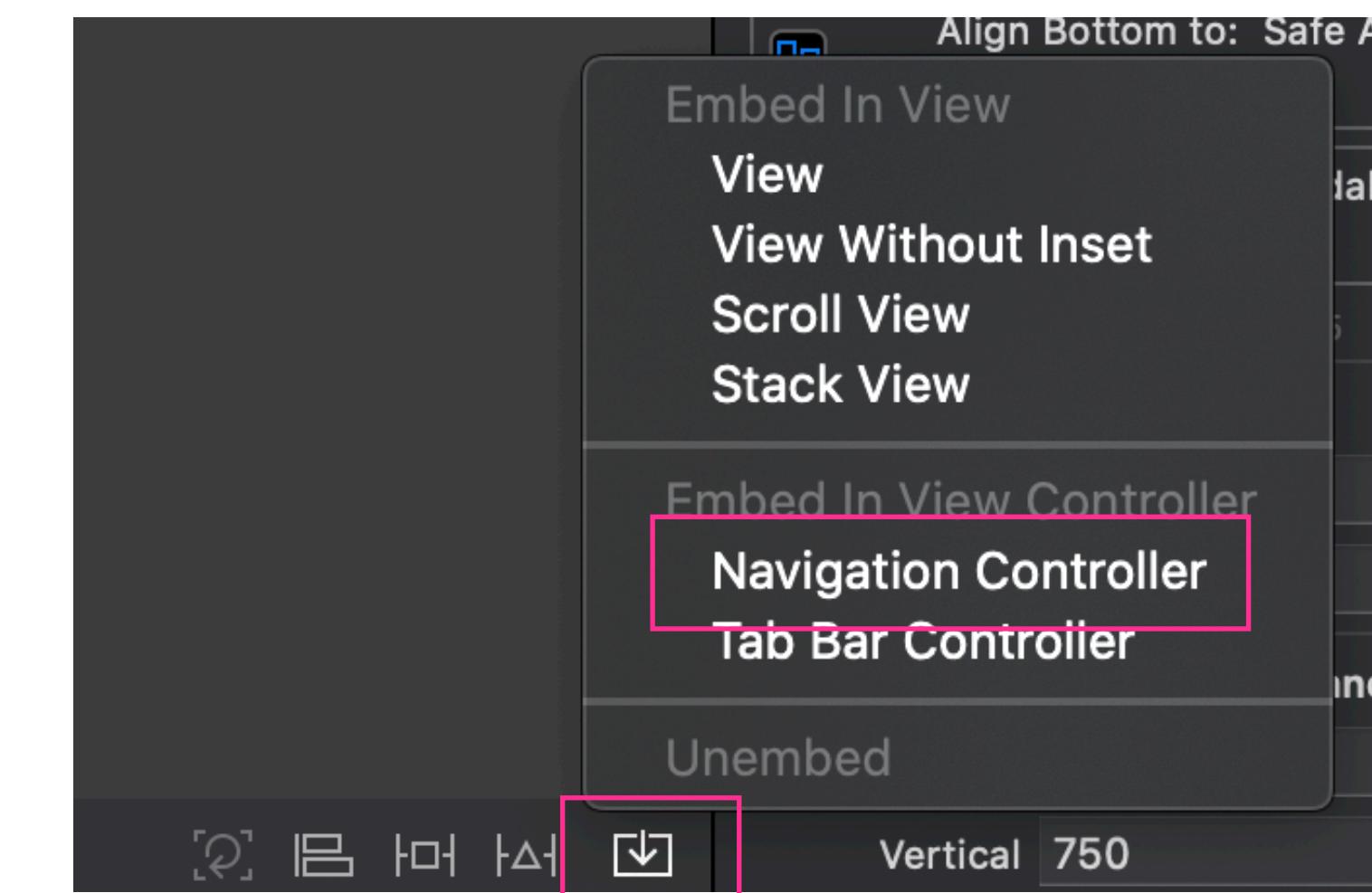
1

UIButtonを2つ配置

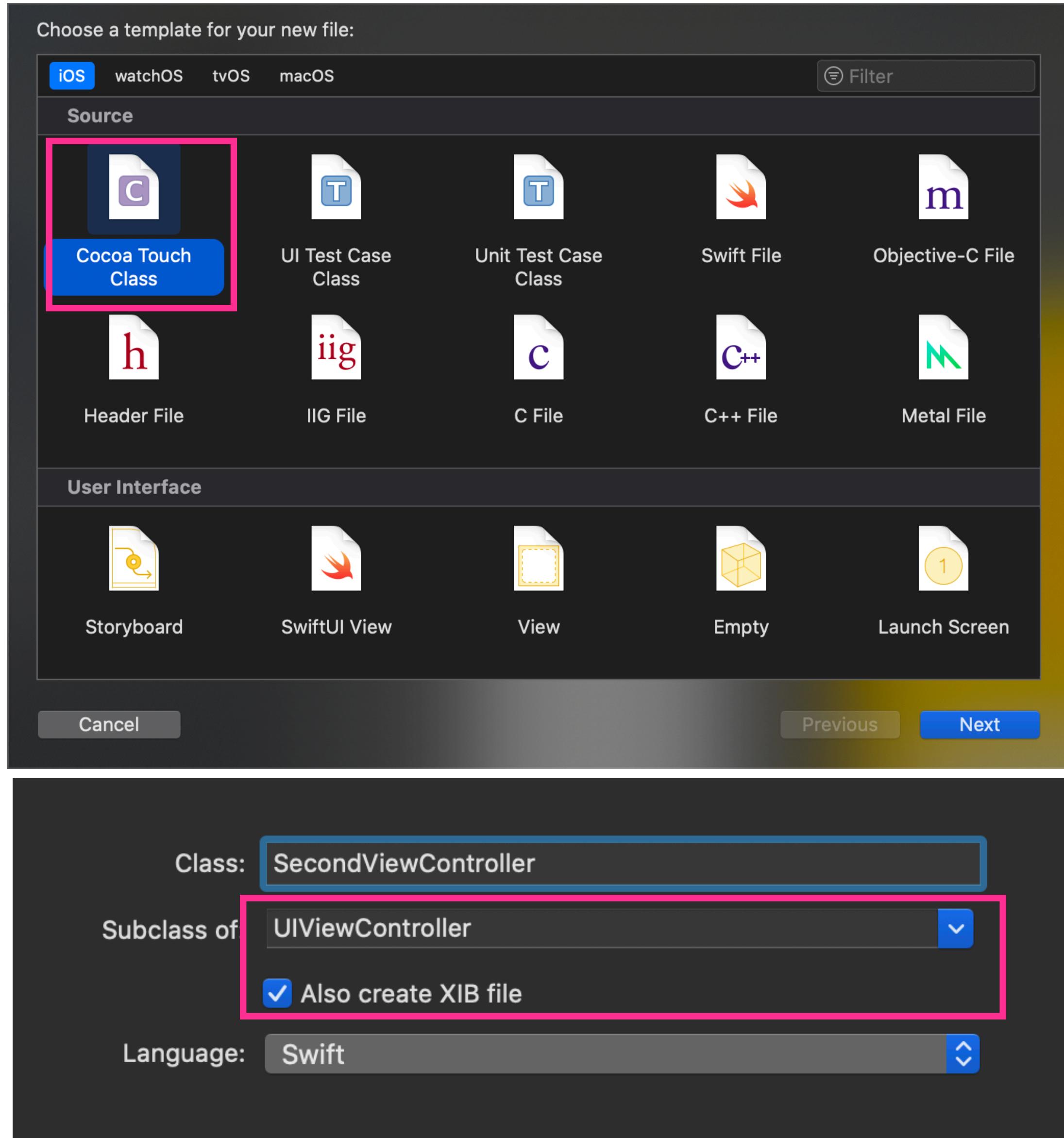
- Width 300pt, Height 60pt
- 左右中心
- 画面下から 70pt
- ボタン間は 48pt

2

Navigation Controllerを追加



Codeで画面遷移(2/4)



1

File追加を選択

- メニューから[File]-[New]-[File]
- もしくは[Command] + [n]

2

Cocoa Touch Class を選択

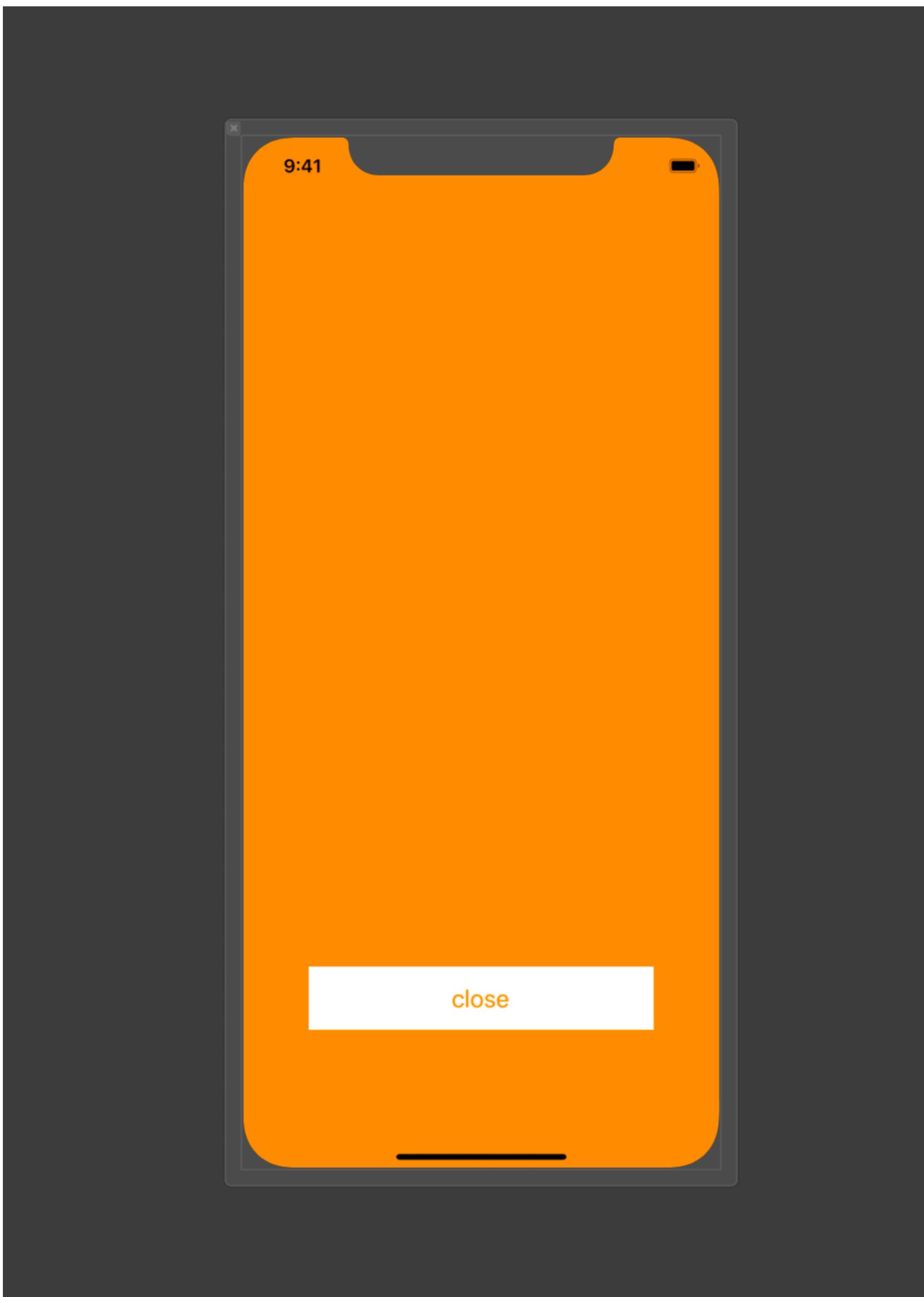
- Appleが用意した機能を引き継ぐとき
- UIViewControllerを選択

3

"Also create XIB file"をON

- XIBファイルはstoryboardのように画面UIを配置できる(storyboardの機能縮小版)

Codeで画面遷移(3/4)



1

遷移先の画面：UIButtonを1つ配置

- Width 300pt, Height 60pt
- 左右中心
- 画面下から 70pt
- ボタン間は 48pt

Codeで画面遷移(4/4)

1

viewController を生成

2

Modal: "self.present()" で画面遷移

- 遷移先で "self.dismiss()" で閉じる。

3

Push: "navigationController?.pushViewController()" で画面遷移

- 遷移先で "navigationController?.popViewController()" で戻る。

```
17  
18 @IBAction func tappedModalButton(_ sender: Any) {  
19     let vc = SecondViewController()  
20     self.present(vc, animated: true, completion: nil)  
21 }  
22  
23 @IBAction func tappedPushButton(_ sender: Any) {  
24     let vc = ThirdViewController()  
25     navigationController?.pushViewController(vc, animated: true)  
26 }  
27 }  
28  
29
```

学習のまとめ

1

コードで画面遷移した方が管理上ラクになります

- storyboardの遷移は直感的で楽だけど、後々困ることになる

2

新しいファイルを"xib"ファイルとペアで作成

- "xib"でUIを配置して、viewControllerで動きを制御する

3

modalとpushの2種類の画面遷移

- Modal: self.present()とself.dismiss()
- Push: navigationControllerのpush()とpop()

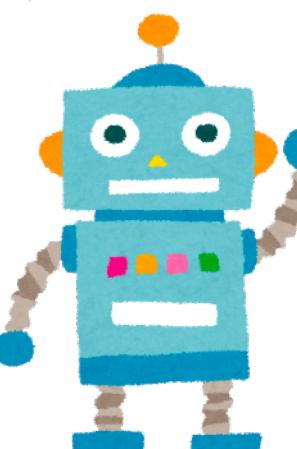
イメージで感覚をつかむ

○○ViewController

- ViewControllerが生成される
(起動時や別の画面から
ViewControllerがつくられる)

別のところから
「いでよ！
○○ViewController！」

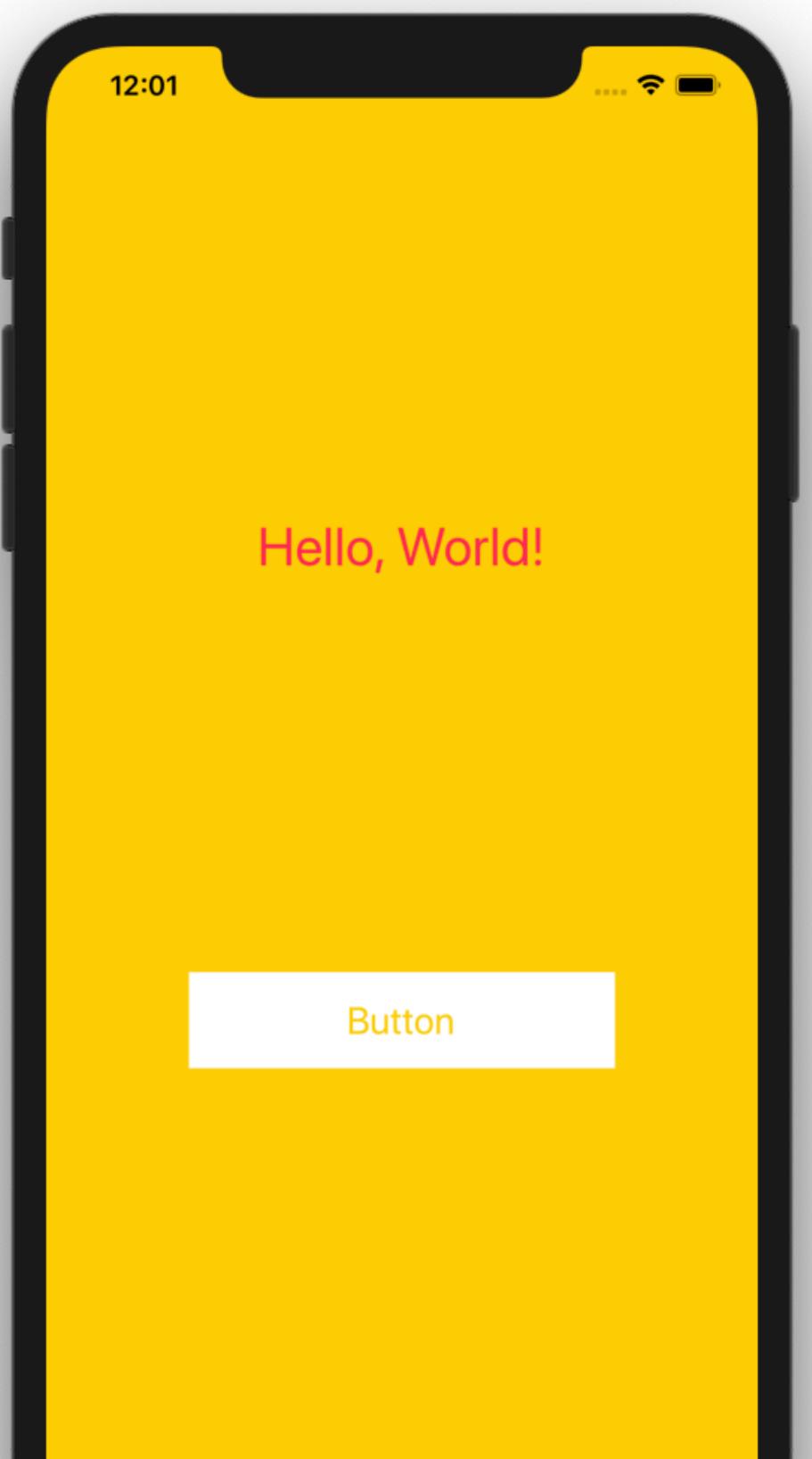
呼ばれたので画面表示するぞ！
お、○○Storyboardと繋がってるな！
(@IBOutlet)



○○ViewController

○○Storyboard

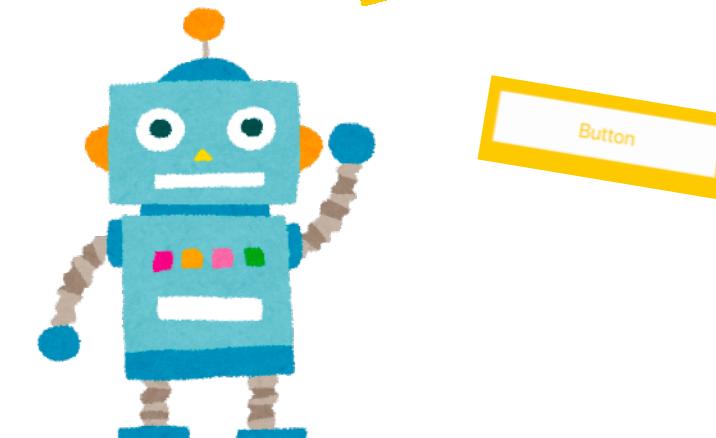
- 関連付けされているStoryboardから
アプリ画面の”初期配置”を読み込む



○○ViewController

- 中身を読みこんだ後に、
func viewDidLoad() が動く
- その後、レイアウトして描画する
(読み込み→レイアウト→描画の順)

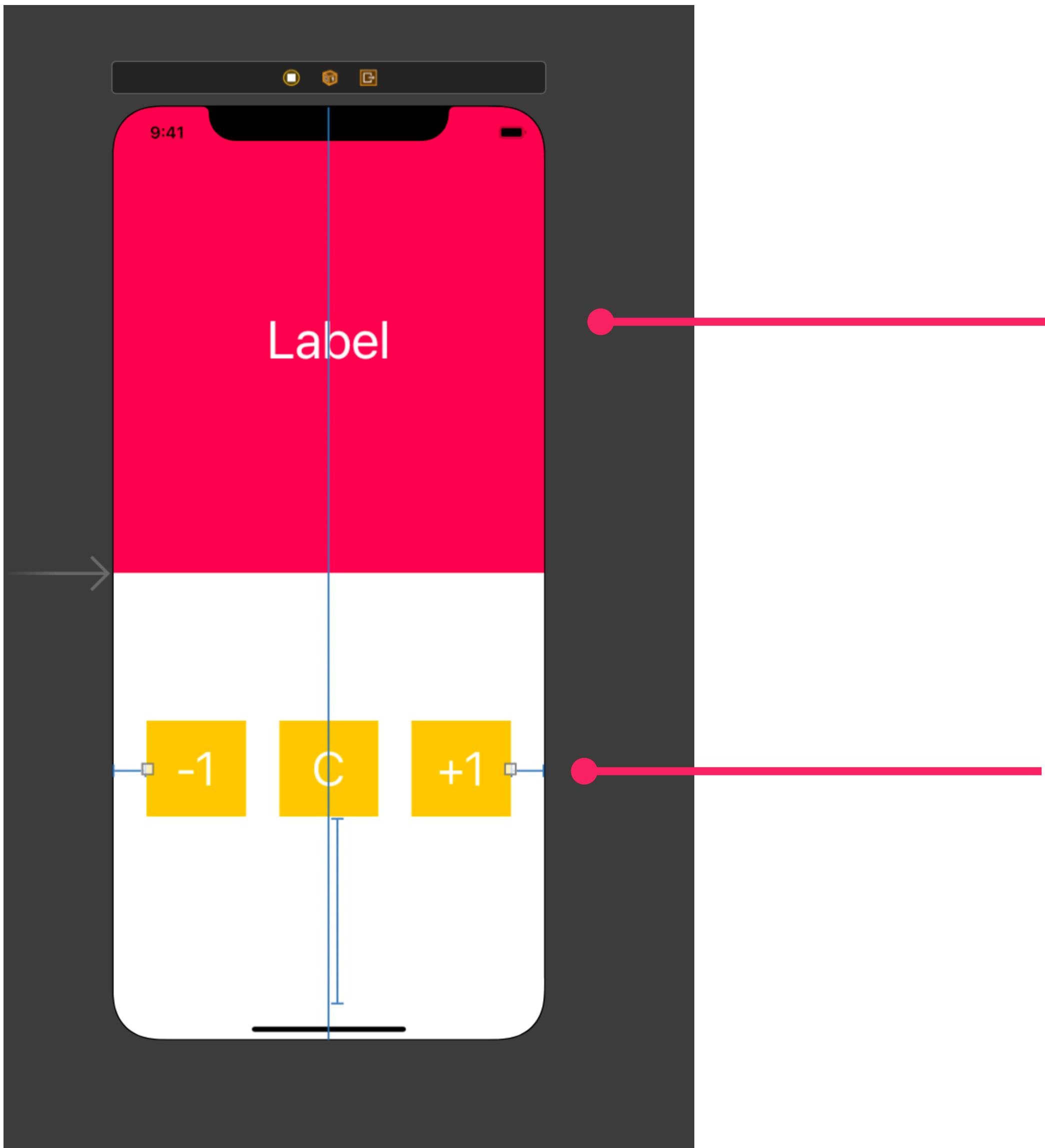
画面を表示するねー。
ボタンタップ時とかも
自分が管理するねー



○○ViewController

＜やってみよう＞
カウントアップアプリを作る

1. 画面UIを作成



UILabel
上左右 画面全体へぴったり
Height: 画面全体の50%

UIButton
真ん中：左右中心
Height/Width: 92pt
画面下から180pt
左右ボタン：真ん中ボタンから32pt

2. Outlet接続・Action接続

```
8  
9 import UIKit  
10  
11 class ViewController: UIViewController {  
12  
13     // カウント表示  
14     @IBOutlet weak var countLabel: UILabel!  
15  
16     override func viewDidLoad() {  
17         super.viewDidLoad()  
18         // Do any additional setup after loading the view.  
19     }  
20  
21     // クリアボタンがタップされたとき  
22     @IBAction func tappedClearButton(_ sender: Any) {  
23     }  
24  
25     // +1ボタンがタップされたとき  
26     @IBAction func tappedPlusButton(_ sender: Any) {  
27     }  
28  
29     // -1ボタンがタップされたとき  
30     @IBAction func tappedMinusButton(_ sender: Any) {  
31     }  
32  
33 }  
34  
35 }
```

[Outlet接続]

- カウントを表示するラベル

[Action接続]

- Clearボタンをタップしたとき
- +1ボタンをタップしたとき
- -1ボタンをタップしたとき

3. ロジックの実装

```
9 import UIKit
10
11 class ViewController: UIViewController {
12
13     // カウント表示
14     @IBOutlet weak var countLabel: UILabel!
15
16     // 現在の状態
17     var count: Int = 0
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         // Do any additional setup after loading the view.
22         displayCount()
23     }
24
25     // クリアボタンがタップされたとき
26     @IBAction func tappedClearButton(_ sender: Any) {
27         count = 0
28         displayCount()
29     }
30
31     // +1ボタンがタップされたとき
32     @IBAction func tappedPlusButton(_ sender: Any) {
33         count += 1
34         displayCount()
35     }
36
37     // -1ボタンがタップされたとき
38     @IBAction func tappedMinusButton(_ sender: Any) {
39         count += -1
40         displayCount()
41     }
42
43     // 変数countの値をUILabelで表示
44     func displayCount() {
45         countLabel.text = String(count)
46     }
47
48 }
```

1. 現在の状態を表すcount変数を定義

2. クリア、+1、-1の場合のcount変数の変化を書く

3. count変数の値をUILabelに反映

4. リファクタリング(応用編)

変数定義後のdidSet

```
// 現在の状態
var count: Int = 0 {
    didSet {
        print("didSetが呼ばれてるよ😡")
        displayCount()
    }
}
```

変数の値が変更されたときに、
didSet {--} の{}内が呼ばれる仕組み。

ある状態を管理して、
その状態が変わったときに○○したい！
というときに便利な書き方。

似たような書き方でwillSet{}もあります。
気になる人は調べてみてください。

データをアプリ内に残しておきたいとき：UserDefaults

UserDefaultsは[キー：値]の形式でアプリ内にデータを保持できる仕組み。

アプリ起動時にデフォルトの状態や動作方法を決定するためによく使います。

(例：初回起動時にはチュートリアル画面を出したい。`didDisplayHome = true`)

書き込み

```
UserDefaults.standard.set({値}, forKey: {キー})
```

読み出し

```
UserDefaults.standard.object(forKey: {キー})
UserDefaults.standard.string(forKey: {キー})
UserDefaults.standard.array(forKey: {キー})
UserDefaults.standard.dictionary(forKey: {キー})
UserDefaults.standard.data(forKey: {キー})
UserDefaults.standard.stringArray(forKey: {キー})
UserDefaults.standard.integer(forKey: {キー})
UserDefaults.standard.float(forKey: {キー})
UserDefaults.standard.double(forKey: {キー})
UserDefaults.standard.bool(forKey: {キー})
UserDefaults.standard.url(forKey: {キー})
```

5. UserDefaultsの追加

```
// 現在の状態
var count: Int = 0 {
    didSet {

        print("didSetが呼ばれてるよ! countの値は \(count)、 oldValueの値は \(oldValue)")
        UserDefaults.standard.set(count, forKey: userDefaultsCountKey)
        displayCount()
    }
}

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    count = UserDefaults.standard.integer(forKey: userDefaultsCountKey)
    print("count:", count)
    displayCount()
}
```

1. countが変更されたときに didSet内で UserDefaultsへ保存
2. viewDidLoadで画面表示前に UserDefaultsから読み込み

Class って何？

Swiftとよく出てくるClassを理解しよう

Classとは？

変数と関数をひとまとめにしたカタマリ

```
class Person {  
    let name: String  
    let age: Int  
    func printMyname() {  
        print("私の名前は\((name)です")  
    }  
    init(name: String, age: Int) {  
        self.name = name  
        self.age = age  
    }  
}
```



変数(プロパティ)



関数(メソッド)



変数に値を入れるための
仕組み

Classとインスタンス

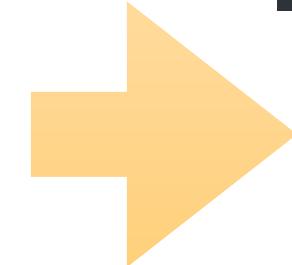
Classは設計図。利用するときはインスタンス化して利用する。

Class

```
class Person {  
    let name: String  
    let age: Int  
    func printMyname() {  
        print("私の名前は\((name)です")  
    }  
    init(name: String, age: Int) {  
        self.name = name  
        self.age = age  
    }  
}
```

インスタンス

```
let kameyama = Person(name: "naoki", age: 34)  
kameyama.name  
kameyama.age  
kameyama.printMyname()  
|
```



- インスタンスはPerson()
- 変数.name (プロパティ)
- 変数.printMyname() (メソッド)

- UIButton, UILabelはAppleが用意したClass

- Outlet接続は裏でインスタンス化している

継承

classを継承して新しい子Classを作ることができる。

継承前のプロパティとメソッドはそのまま使える。

Class

```
class Person {  
    let name: String  
    let age: Int  
    func printMyname() {  
        print("私の名前は\((name)です")  
    }  
    init(name: String, age: Int) {  
        self.name = name  
        self.age = age  
    }  
}
```

継承

```
class Yusha: Person {  
    let hp: Int  
    let mp: Int  
    func attack(enemy: Person) {  
        print("enemyへの攻撃")  
    }  
    init(hp: Int, mp: Int, name: String, age: Int) {  
        self.hp = hp  
        self.mp = mp  
        super.init(name: name, age: age)  
    }  
}
```

継承元のプロパティ
メソッドをそのまま利用できる

```
let yoshihiko = Yusha(hp: 32, mp: 10, name: "ヨシヒコ", age: 32)  
yoshihiko.hp  
yoshihiko.name  
yoshihiko.printMyname()
```

ソースコードの書く位置

```
import UIKit

class FirstViewController: UIViewController {

    @IBOutlet weak var textLabel: UILabel!

    // 変数はどこでも定義できる
    var count: Int = 0

    // class直下のこの位置では初期値を入れないとエラー
    // var value: Int

    // printとかtextLabel.text = "hogehoge"もこの場所ではできない。funcの中でできる。
    // この上にUIパーツを配置する(推奨)

    // viewの要素が読み込まれるとこのfuncが動く(appleが決めた仕組み)
    override func viewDidLoad() {
        super.viewDidLoad()
        print("ここはviewDidLoad")
        textLabel.text = "Viewが読み込まれたよ"
    }

    // この下にUIのActionを配置する(推奨)

    @IBAction func tapButton(_ sender: Any) {
        textLabel.text = "変化したよ"
        textLabel.textColor = .blue
    }
}
```

class直下の位置にはUIパーツや、
var や letで定義した変数・定数が入る

class直下の位置にはそれ以外のprintや
count = 1などの文を入れるとエラー

上記のUIパーツや変数・定数の初期値が読み込まれた後に **func viewDidLoad()** が呼ばれる
(上から順番にコードが動く訳ではない！)

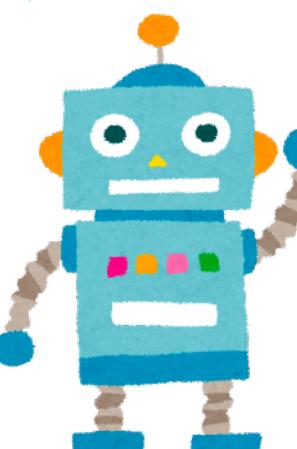
イメージで感覚をつかむ

○○ViewController

- ViewControllerが生成される
(起動時や別の画面から
ViewControllerがつくられる)

別のところから
「いでよ！
○○ViewController！」

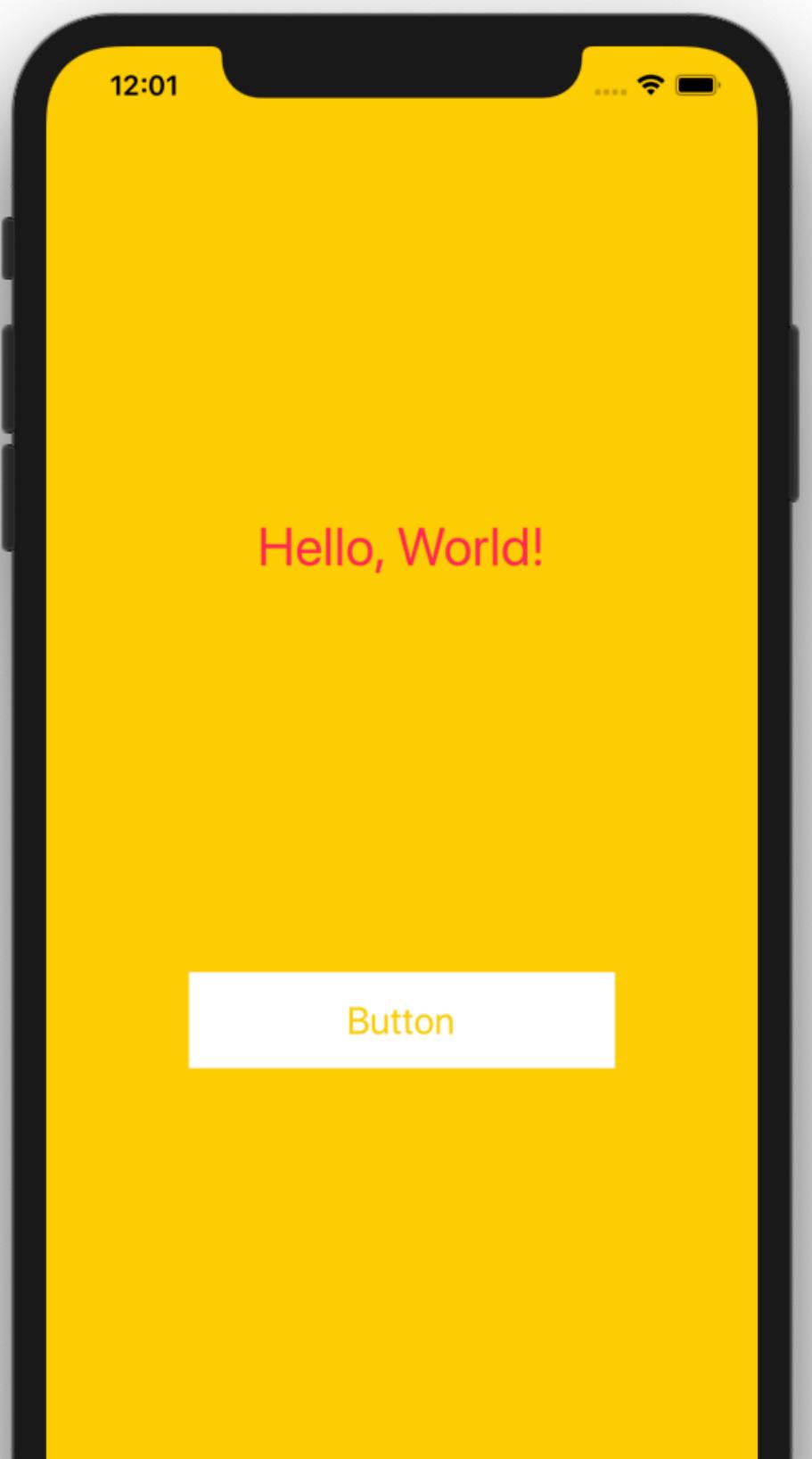
呼ばれたので画面表示するぞ！
お、○○Storyboardと繋がってるな！
(@IBOutlet)



○○ViewController

○○Storyboard

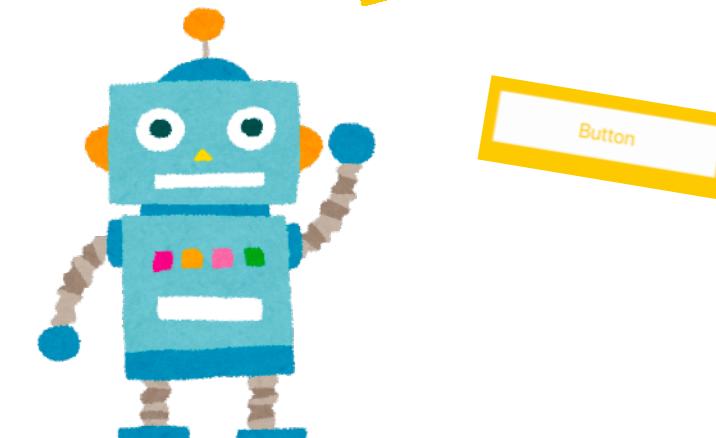
- 関連付けされているStoryboardから
アプリ画面の”初期配置”を読み込む



○○ViewController

- 中身を読みこんだ後に、
func viewDidLoad() が動く
- その後、レイアウトして描画する
(読み込み→レイアウト→描画の順)

画面を表示するねー。
ボタンタップ時とかも
自分が管理するねー



○○ViewController

さいごに

今日のXcodeショートカット！

1

[Command] + [Shift] + [f]

- ・ソースコード全体から検索

2

[Command] + [f]

- ・ファイル内から検索

自分が苦手なので
一緒にクセ付けします。。



次週予告！

1

gitを使ってみよう

- gitの仕組みを学習
- GitHubにpushしてみよう(課題提出方法)

2

通知を使ったアプリを作ってみよう

- delegateの仕組みを知ろう

【本日の課題】

お題「ストップウォッチアプリ」

(最低限の内容)

- start/stop/reset機能

※ラップ機能、保存機能、○秒になったら何かする・・・等

(提出方法)

課題の提出はプロジェクトをzip圧縮して

今までと同じようにフォームから提出して下さい

本日はお疲れ様でした。

