[iOS]

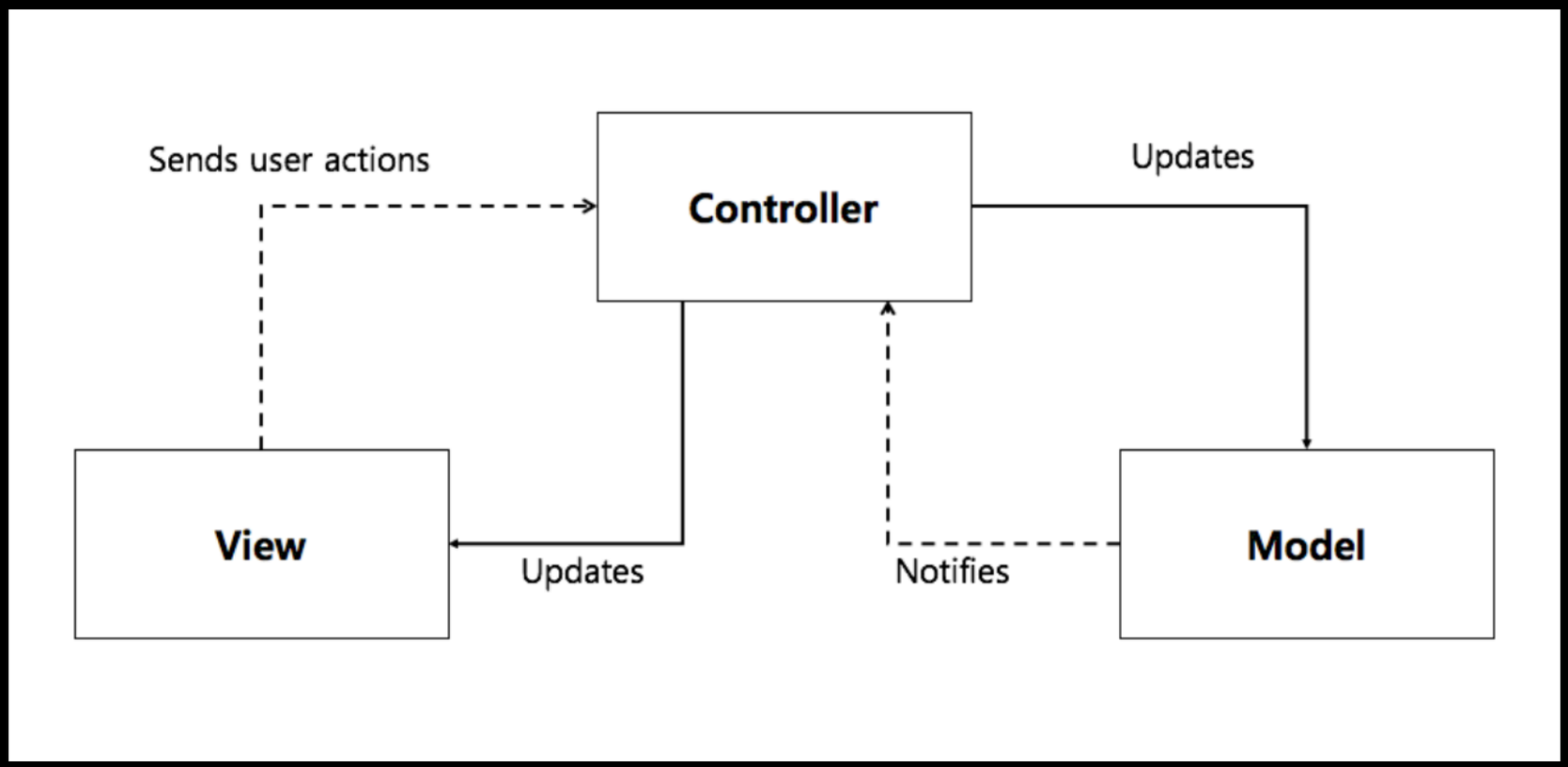
MVC, MVP, MVVM

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Traditional MVC

Traditional MVC



Model : Data(데이터) + Status(상태) + Logic(로직)을 담당
View : Model을 표현하고 책임지는 객체
Controller : Model과 View를 연결해주고 관리하는 객체

Traditional MVC

장점 :

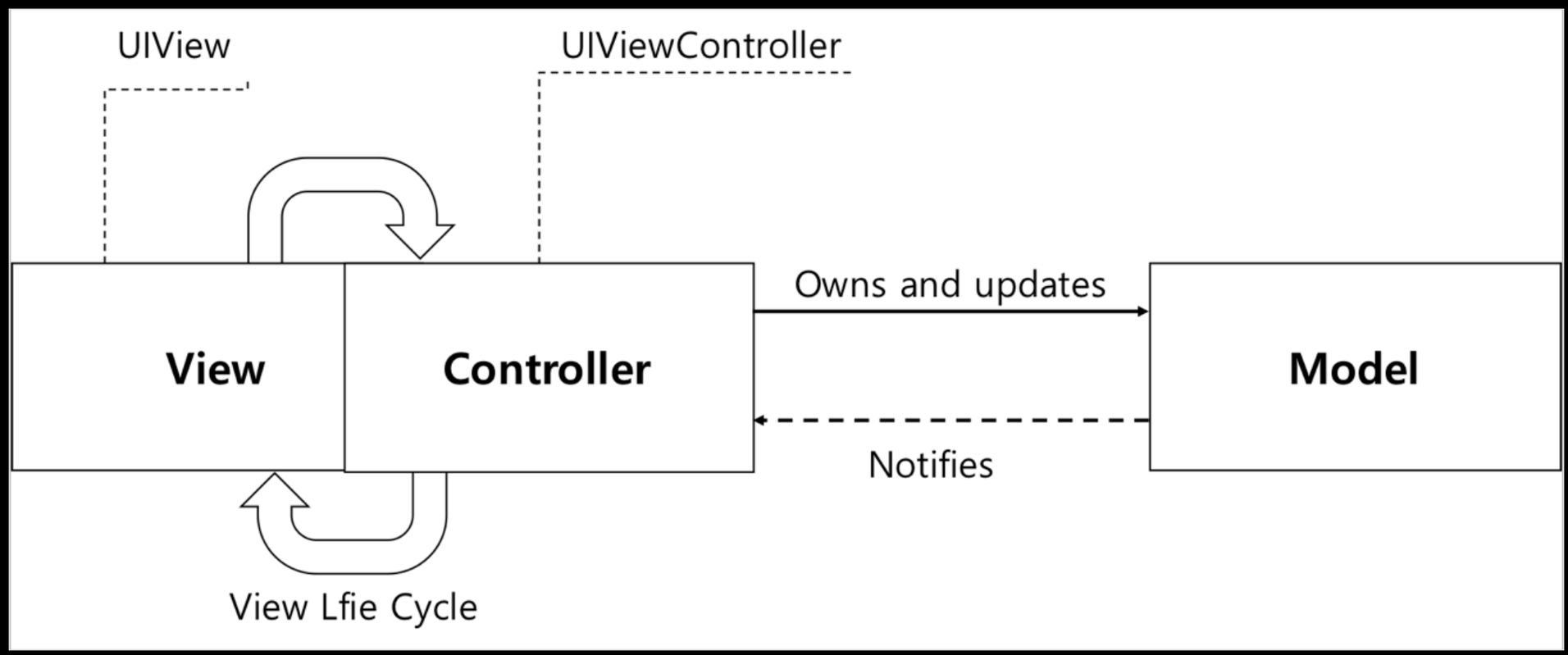
- Model과 View를 확실하게 분리시켜준다.
- Model 이 어디에도 종속되지 않아 쉽게 모델을 테스트 가능

단점 :

- Controller가 View에 따라서 깊에 종속되어 테스트가 제한
- View가 변경되면 Controller도 변경이 필요
- Controller가 커지고 시간이 지나면 유지보수가 어려움
- 최신 iOS 개발에는 부적합해보임

Apple's MVC

Apple's MVC



Model : Data(데이터) + Status(상태) + Logic(로직)을 담당
View Controller : View가 Model을 표현하는 역할을 주로 하며 View Controller와 연결되어 대부분 View Controller가 모든 처리를 담당

Apple's MVC

장점 :

- 전통적인 MVC의 장점과 비슷
- View와 Controller가 사실상 하나라서 코드 라인이 적다
- 개발 코드가 적어서 익숙해지면 개발 속도가 빠르다

단점 :

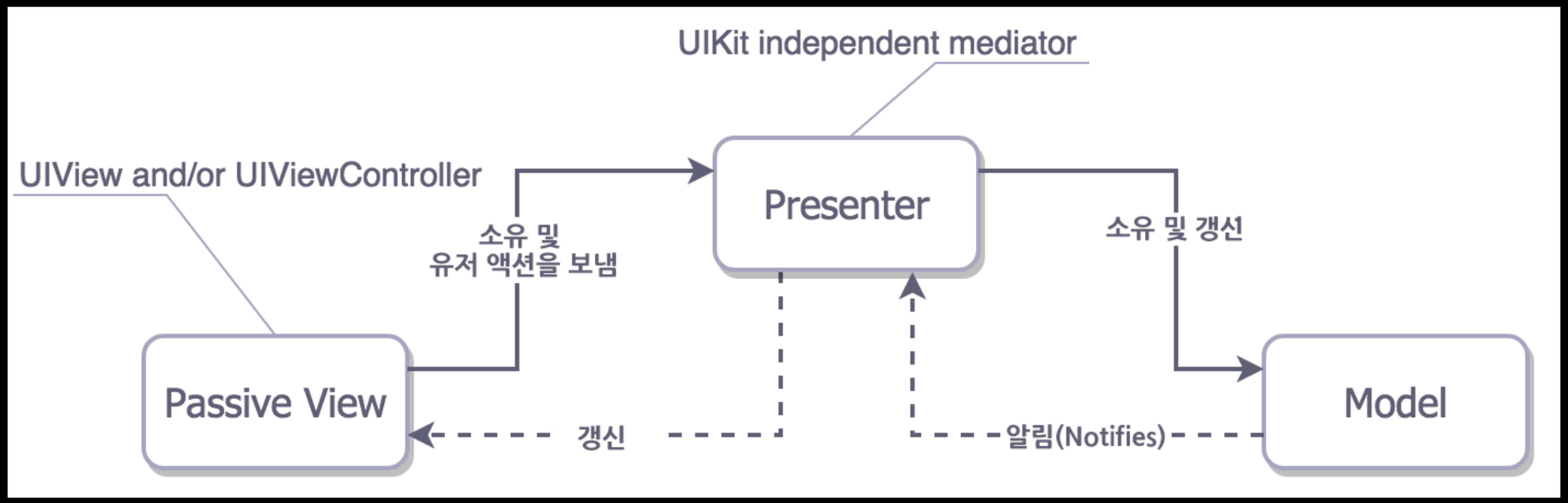
- 전통적인 MVC의 비해 View와 Contrller가 하나로 합쳐져서 View와 Controller의 역할이 모호하며 Contrller가 좀더 커질 수 있음
- 사실상 M + VC의 개념이다
- View 라이프 사이클 및 테스트를 위한 View 작성이 어려움

Sample Code : MVC

```
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

class GreetingViewController : UIViewController { // View + Controller
    var person: Person!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
    }

    func didTapButton(button: UIButton) {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.greetingLabel.text = greeting

    }
    // layout code goes here
}
// Assembling of MVC
let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
view.person = model;
```

Cocoa MVP

Cocoa MVP



Model : Data(데이터) + Status(상태) + Logic(로직)을 담당
(Passive) View : Presenter를 소유하며 액션을 보내주는 객체
Presenter : Model을 소유하고 모델이 갱신되면 View 에 전달

Cocoa MVP

장점 :

- Model 및 Presenter에 거의 책임을 분리하여 View는 간결해짐
- View와 별개로 비지니스 로직만 테스트 가능
- 전반적으로 MVC보다 깔끔한 코딩이 가능

단점 :

- 일반적인 MVC보다 명확한 구분으로 인한 코드가 길다
- 크지 않은 시스템에서는 비효율적인 구조가 될 수 있다
- Presenter가 커지면 기존 MVC 처럼 유지보수가 어렵다
- 경우에 Presenter를 여러개로 분리, 관리하기도 쉽지 않게된다

Sample Code : MVP

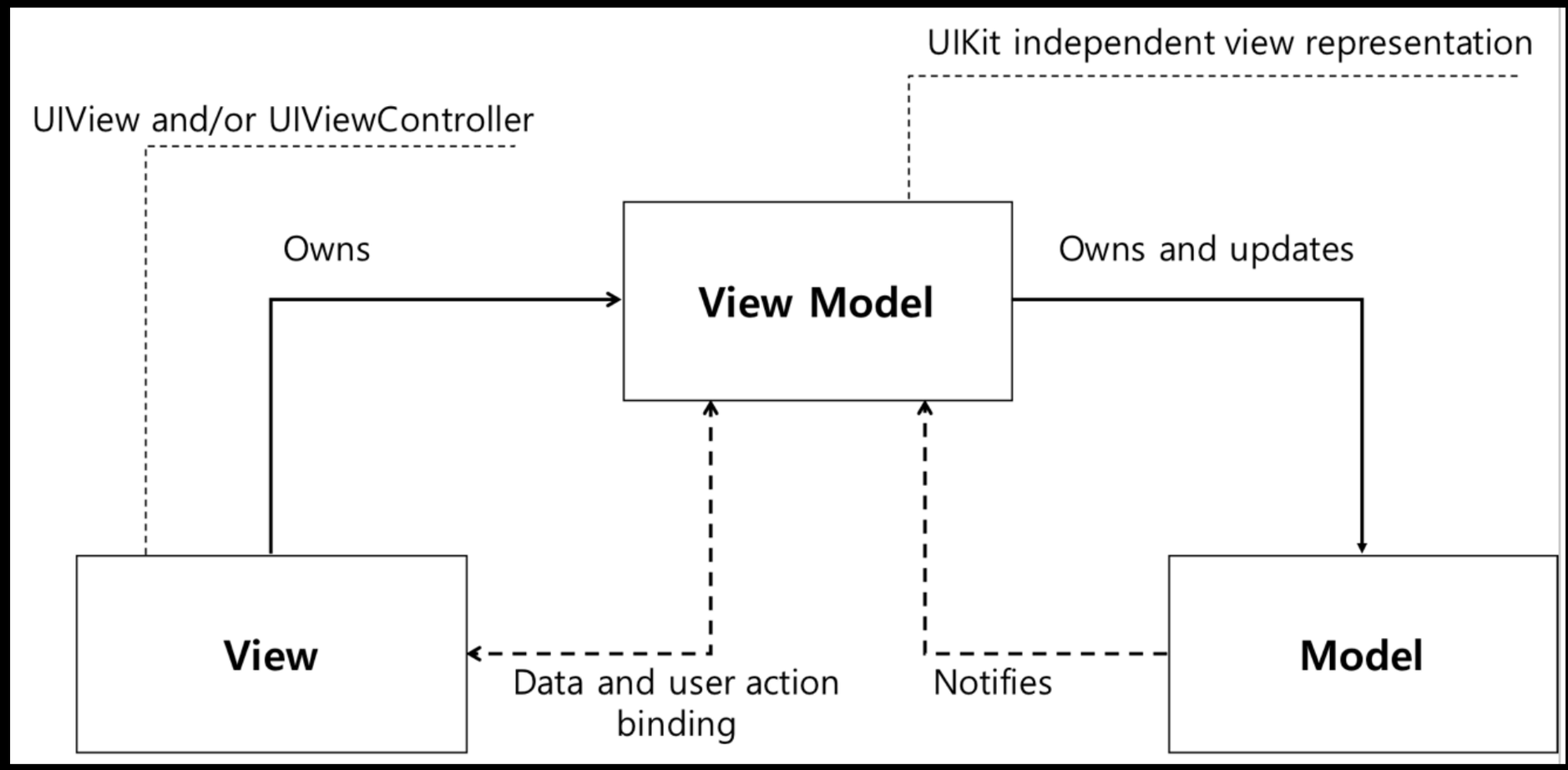
```swift
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

protocol GreetingViewPresenter {
    init(view: GreetingView, person: Person)
    func showGreeting()
}

class GreetingPresenter : GreetingViewPresenter {
    unowned let view: GreetingView
    let person: Person
    required init(view: GreetingView, person: Person) {
        self.view = view
        self.person = person
    }
    func showGreeting() {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.view.setGreeting(greeting)
    }
}
```

Sample Code : MVP

```
class GreetingViewController : UIViewController, GreetingView {
    var presenter: GreetingViewPresenter!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
    }

    func didTapButton(button: UIButton) {
        self.presenter.showGreeting()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // layout code goes here
}
// Assembling of MVP
let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
let presenter = GreetingPresenter(view: view, person: model)
view.presenter = presenter
```

Cocoa MVVM

Cocoa MVVM



Model : MVC, MVP에서의 Model와 같음

View : View Controller가 View의 역할을 함

View Model : Model을 소유하고 모델을 직접 갱신한다. 갱신에 대한 이벤트 및 액션을 View에 바인딩하여 전달한다.

Cocoa MVVM

장점 :

- View Model이 View에 대해서 전혀 모르며 테스트가 용이
- View 또한 개별적인 테스트 용이
- MVP보다 깔끔하며 코딩 양이 적다
- View는 View Model과의 바인딩을 통하여 Model의 값의 변경을 쉽게 추적 및 업데이트 할 수 있다.

단점 :

- 기본 MVP의 View보다 MVVM의 View는 책임의 범위가 더 크다
- Reactive 프레임워크를 주로 사용하기 때문에 경우에 따라 디버깅이 어려움
- 간단한 앱의 경우 MVC가 오히려 관리가 편하고 코드가 적음

Sample Code : MVVM

```swift
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

protocol GreetingViewModelProtocol: class {
    var greeting: String? { get }
    var greetingDidChange: ((GreetingViewModelProtocol) -> ())? { get set } // function to
    init(person: Person)
    func showGreeting()
}

class GreetingViewModel : GreetingViewModelProtocol {
    let person: Person
    var greeting: String? {
        didSet {
            self.greetingDidChange?(self)
        }
    }
    var greetingDidChange: ((GreetingViewModelProtocol) -> ())?
    required init(person: Person) {
        self.person = person
    }
    func showGreeting() {
        self.greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
    }
}
```

Sample Code : MVVM

```
class GreetingViewController : UIViewController {
    var viewModel: GreetingViewModelProtocol! {
        didSet {
            self.viewModel.greetingDidChange = { [unowned self] viewModel in
                self.greetingLabel.text = viewModel.greeting
            }
        }
    }
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self.viewModel, action: "showGreeting", forContro
    }
    // layout code goes here
}
// Assembling of MVVM
let model = Person(firstName: "David", lastName: "Blaine")
let viewModel = GreetingViewModel(person: model)
let view = GreetingViewController()
view.viewModel = viewModel
```

References

References

[1] iOS Architecture Patterns : http://labs.brandi.co.kr/2018/02/21/kimjh.html

[2] (번역) iOS 아키텍쳐 패턴들 : https://blog.canapio.com/43

[3] [디자인패턴] MVC, MVP, MVVM 비교 : https://beomy.tistory.com/43

[4] MVC, MVP, MVVM 비교 : https://magi82.github.io/android-mvc-mvp-mvvm/

[5] 안드로이드의 MVC, MVP, MVVM 종합 안내서 : https://academy.realm.io/kr/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/

Thank you!