今更聞けない MVPとMVVMの違い

サポーターズCoLab勉強会: April 5th

Taiki Suzuki / @marty_suzuki

Profile

Taiki Suzuki

Overview

Popular repositories

SAHistoryNavigationViewController
SAHistoryNavigationViewController realizes iOS task manager like UI in UINavigationController.
● Swift ★ 1,515

ReverseExtension
A UITableView extension that enables cell insertion from the bottom of a table view.
● Swift ★ 1,248

SABlurImageView
You can use blur effect and it's animation easily to call only two methods.
● Swift ★ 490

URLEmbeddedView
URLEmbeddedView automatically caches the object that is confirmed the Open Graph Protocol.
● Swift ★ 451

Activity

marty-suzuki

marty_suzuki

CyberAgent, Inc.

モバイルアプリ開発エキスパート養成読本

Android iOS

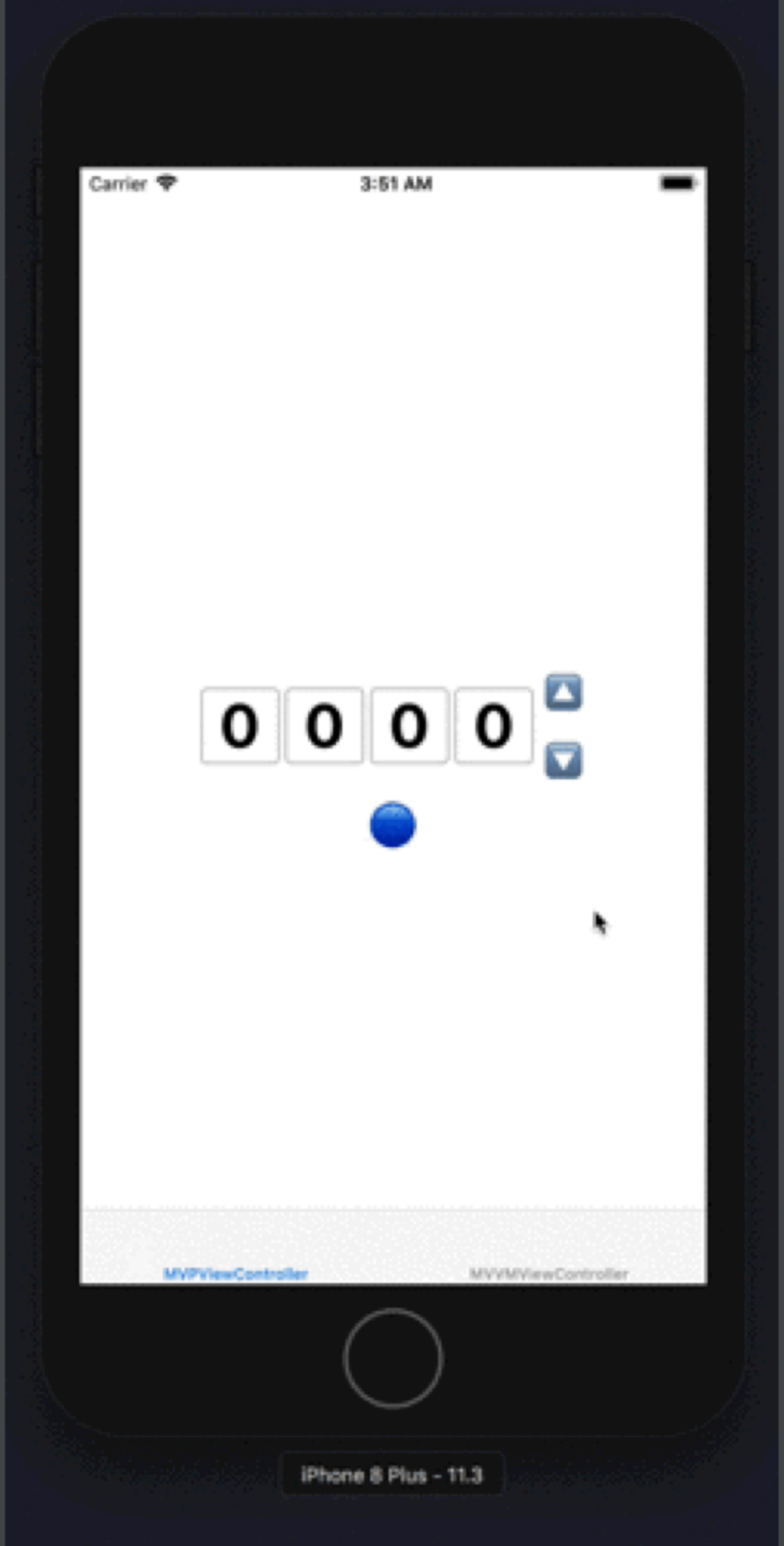
リアクティブプログラミング入門

現場で役立つ設計・開発

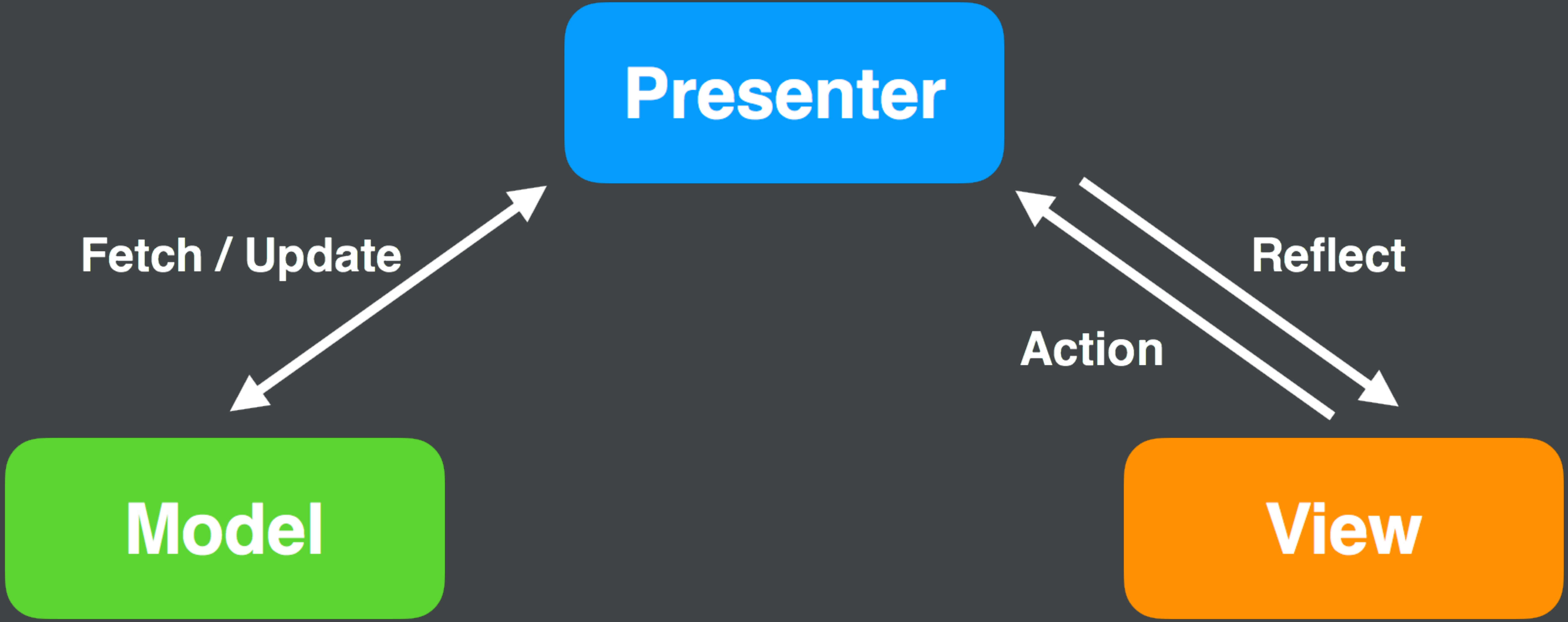
What are differences between MVP and MVVM ?

Tally Counter

数取器



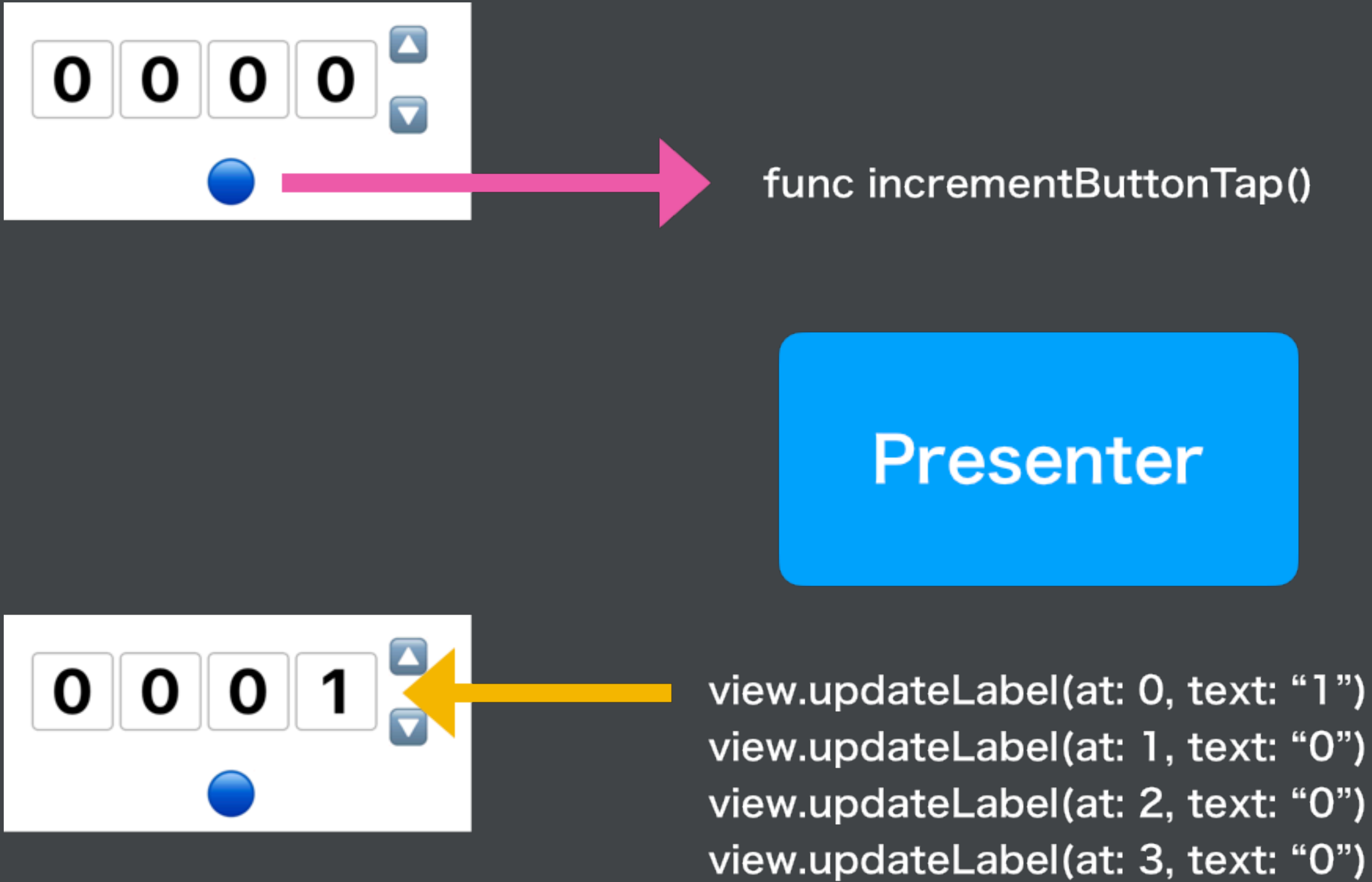
MVP


Presenter
Fetch / Update
Action
Reflect
Model
View

```swift
protocol CounterView: class {
    func updateLabel(at index: Int, text: String)
}
```

```swift
final class MVPViewController: UIViewController, CounterView {
    @IBOutlet private(set) var labels: [UILabel]!
    @IBOutlet private(set) weak var incrementButton: UIButton!
    @IBOutlet private(set) weak var upButton: UIButton!
    @IBOutlet private(set) weak var downButton: UIButton!

    private lazy var presenter = CounterPresenter(numberOfDigits: self.labels.count, view: self)

    override func viewDidLoad() {
        super.viewDidLoad()

        incrementButton.addTarget(presenter, action: #selector(CounterPresenter.incrementButtonTap), for: .touchUpInside)
        upButton.addTarget(presenter, action: #selector(CounterPresenter.upButtonTap), for: .touchUpInside)
        downButton.addTarget(presenter, action: #selector(CounterPresenter.downButtonTap), for: .touchUpInside)
    }

    func updateLabel(at index: Int, text: String) {
        labels[index].text = text
    }
}
```

```swift
final class CounterPresenter {
    private weak var view: CounterView?
    ... // other properties

    init(numberOfDigits: Int, view: CounterView) {
        self.view = view
        ... // other implementations
    }

    @objc func incrementButtonTap() {
        ... // increment implementations
    }

    @objc func upButtonTap() {
        ... // up implementations
    }

    @objc func downButtonTap() {
        ... // down implementations
    }
}
```



```
func incrementButtonTap()
```

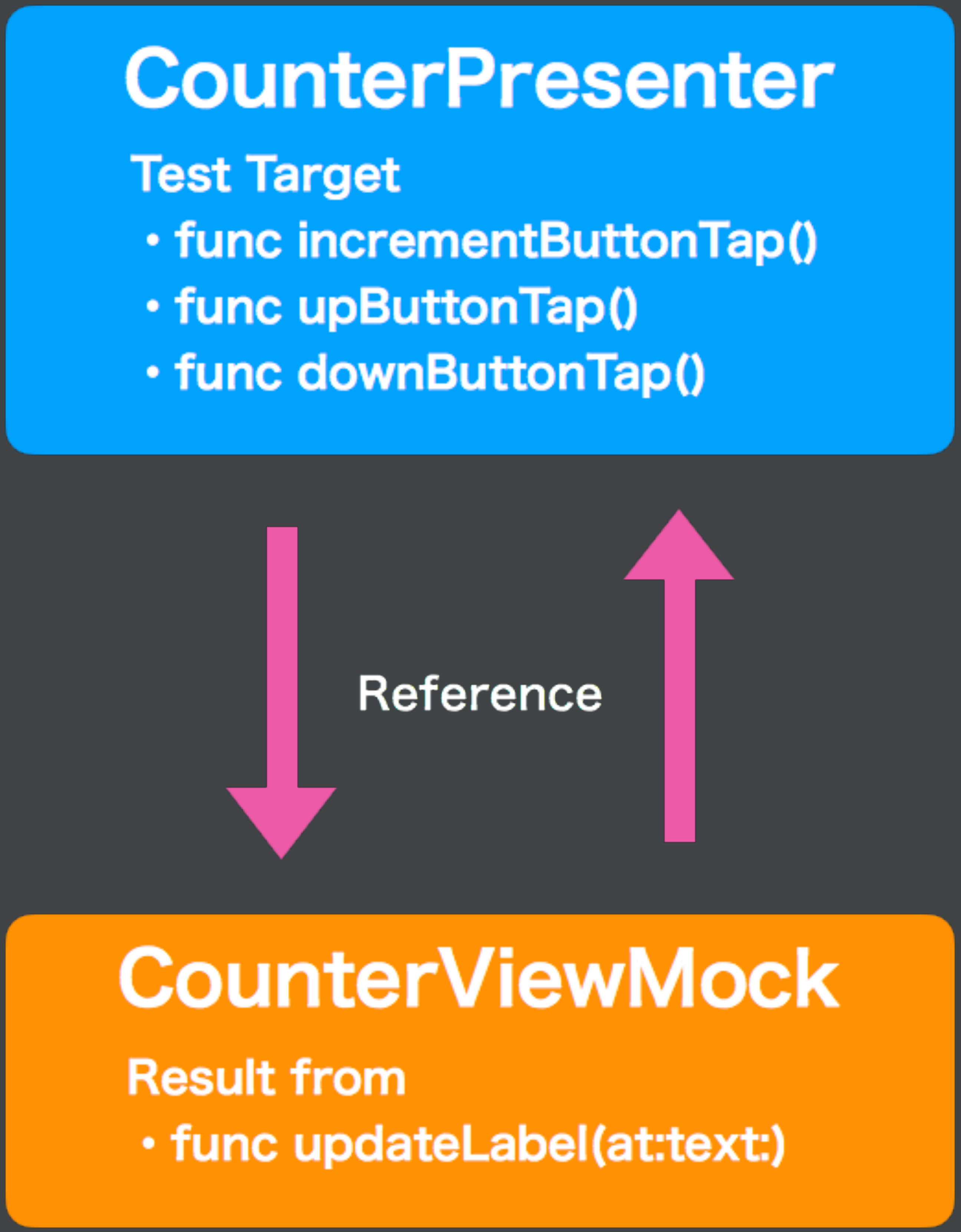
```
view.updateLabel(at: 0, text: “1”)
view.updateLabel(at: 1, text: “0”)
view.updateLabel(at: 2, text: “0”)
view.updateLabel(at: 3, text: “0”)
```

MVP Unit Test

Presenter Unit Test

```swift
final class CounterViewMock: CounterView {
    struct UpdateParameters {
        let index: Int
        let text: String
    }

    private let didCallUpdateLabel: (UpdateParameters) -> ()

    init(didCallUpdateLabel: @escaping (UpdateParameters) -> ()) {
        self.didCallUpdateLabel = didCallUpdateLabel
    }

    func updateLabel(at index: Int, text: String) {
        didCallUpdateLabel(UpdateParameters(index: index, text: text))
    }
}
```

```swift
final class CounterPresenterTestCase: XCTestCase {
    private var counterView: CounterViewMock!
    private var presetner: CounterPresenter!
    private var results: [CounterViewMock.UpdateParameters]!

    override func setUp() {
        super.setUp()

        self.results = []
        let counterView = CounterViewMock() { [weak self] result in
            self?.results.append(result)
        }
        self.presetner = CounterPresenter(numberOfDigits: 4, view: counterView)
        self.counterView = counterView
    }

    ... // other implementations
}
```



CounterPresenter

Test Target

- func incrementButtonTap()
- func upButtonTap()
- func downButtonTap()

CounterViewMock

Result from

- func updateLabel(at:text:)

```swift
final class CounterPresenterTestCase: XCTestCase {
    ... // other implementations

    func testInitialValue() {
        XCTAssertEqual(results.count, 4)
        results.forEach { result in
            XCTAssertEqual(result.text, "0")
        }
    }

    func testIncrementButtonTap() {
        XCTAssertEqual(results.count, 4)
        presetner.incrementButtonTap()

        XCTAssertEqual(results.count, 8)
        XCTAssertEqual(results[4].text, "1")
        XCTAssertEqual(results[5].text, "0")
        XCTAssertEqual(results[6].text, "0")
        XCTAssertEqual(results[7].text, "0")
    }
}
```

UIViewController Unit Test

Depends on … 🤔

```
final class MVPViewController: UIViewController, CounterView {
    ... // other properties

    private lazy var presenter = CounterPresenter(numberOfDigits: self.labels.count, view: self)

    ... //  other implementations
}
```

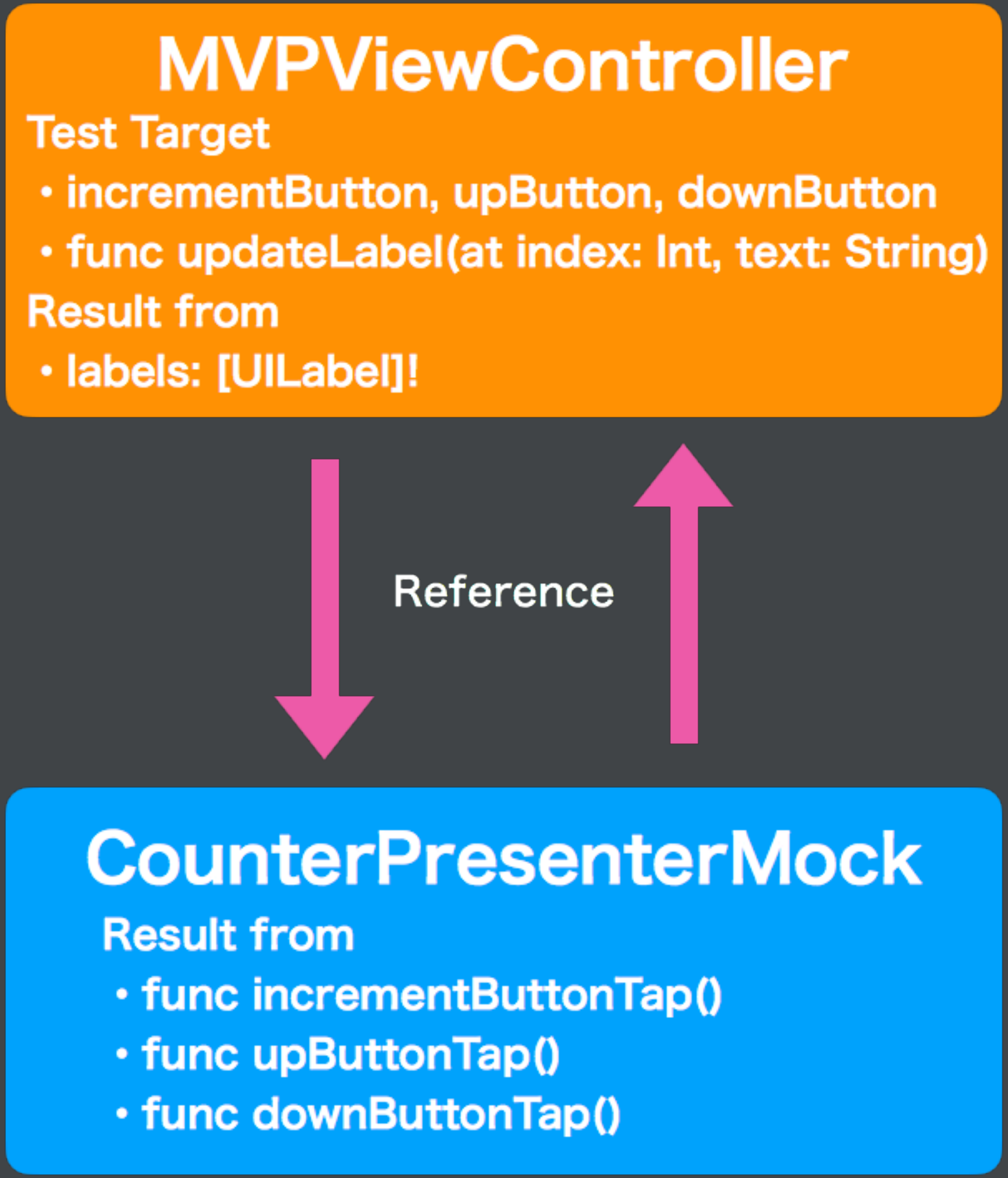
```swift
@objc protocol CounterPresenterType: class {
    init(numberOfDigits: Int, view: CounterView)
    @objc func incrementButtonTap()
    @objc func upButtonTap()
    @objc func downButtonTap()
}

final class CounterPresenter: CounterPresenterType {
    ... // other implementations
}
```

```swift
final class MVPViewController<Presenter: CounterPresenterType>: UIViewController,
                                                                 CounterView {
    ... // other properties

    private lazy var presenter = Presenter(numberOfDigits: self.labels.count,
                                           view: self)


    init() {
        super.init(nibName: "MVPViewController", bundle: nil)
    }

    ... // other implementations
}
```

Ready for Testing 👍

```swift
final class CounterPresenterMock: CounterPresenterType {
    let numberOfDigits: Int
    private(set) weak var view: CounterView?
    private(set) var incrementButtonTapCount: Int = 0
    private(set) var upButtonTapCount: Int = 0
    private(set) var downButtonTapCount: Int = 0

    init(numberOfDigits: Int, view: CounterView) {
        self.numberOfDigits = numberOfDigits
        self.view = view
    }

    @objc func incrementButtonTap() {
        incrementButtonTapCount += 1
    }

    @objc func upButtonTap() {
        upButtonTapCount += 1
    }

    @objc func downButtonTap() {
        downButtonTapCount += 1
    }
}
```

```swift
final class MVPViewControllerTestCase: XCTestCase {
    private var viewController: MVPViewController!
    private var presenter: CounterPresenterMock!

    override func setUp() {
        super.setUp()

        let viewController = MVPViewController<CounterPresenterMock>()
        _ = viewController.view
        self.viewController = viewController
        self.presenter = viewController.presenter
    }

    ... // other implementations
}
```



MVPViewController

Test Target
- incrementButton, upButton, downButton
- func updateLabel(at index: Int, text: String)

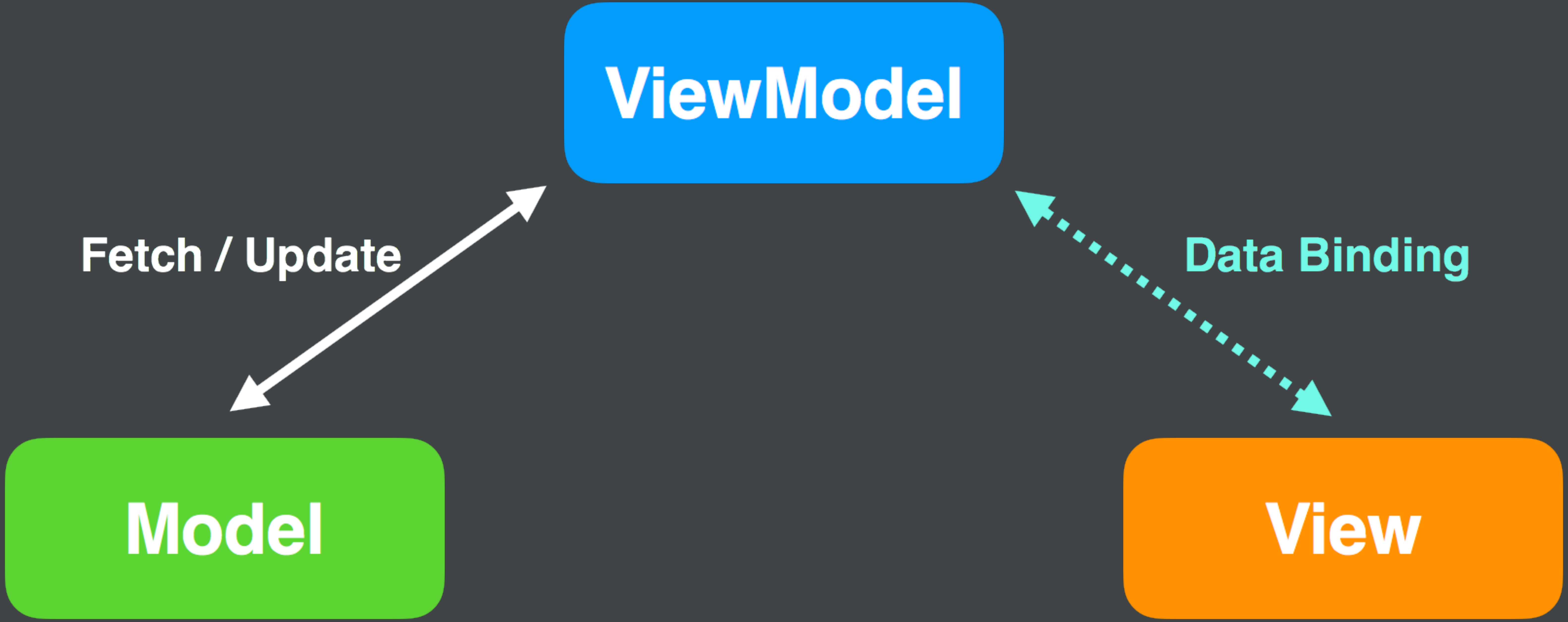
Result from
- labels: [UILabel]!

CounterPresenterMock

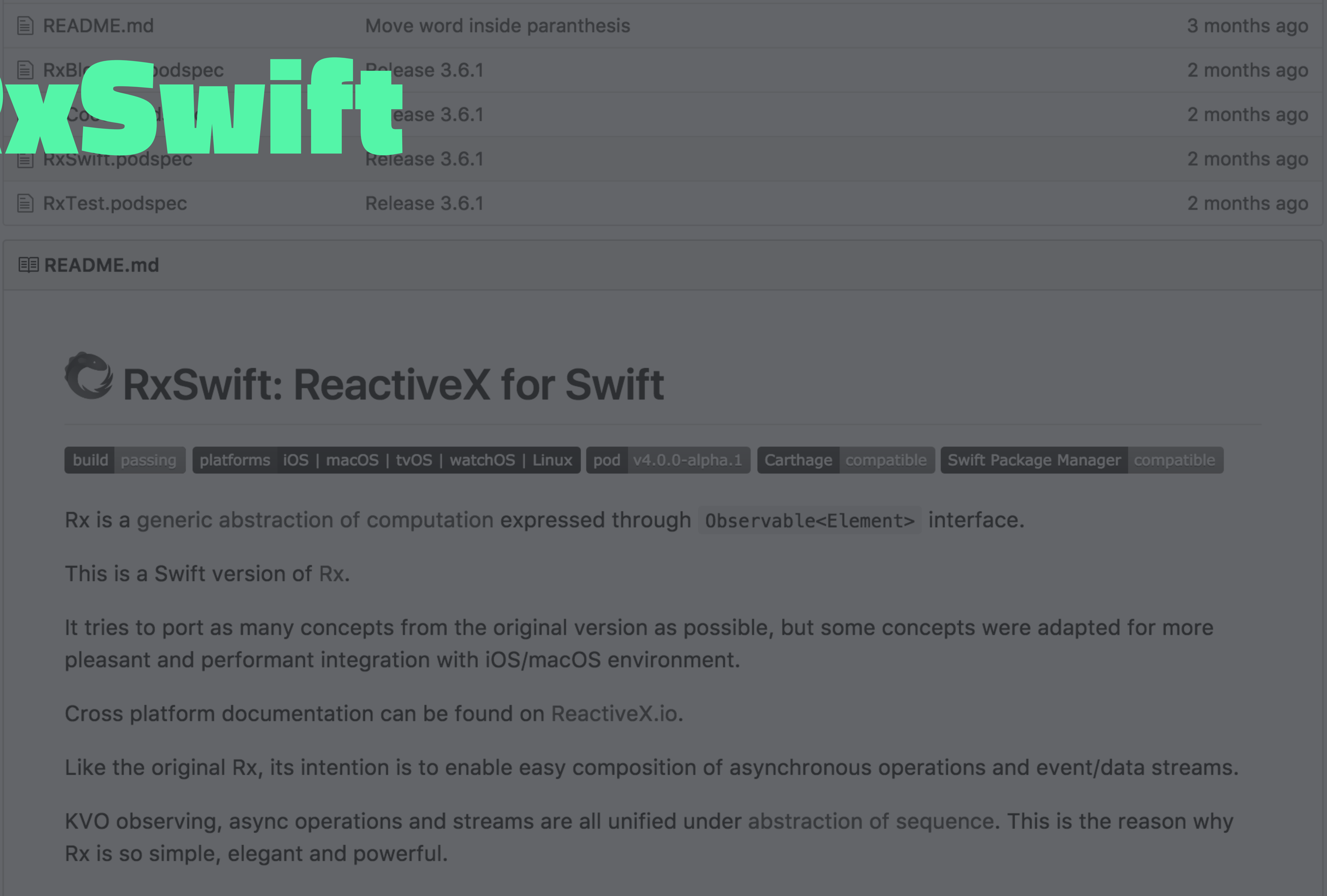
Result from
- func incrementButtonTap()
- func upButtonTap()
- func downButtonTap()

```swift
final class MVPViewControllerTestCase: XCTestCase {
   // other implementations

   func testNumberOfPlaceValues() {
       XCTAssertEqual(viewController.labels.count, presenter.numberOfDigits)
   }

    func testUpdateLabelAtIndex0To1() {
        let index: Int = 0
        XCTAssertNotNil(viewController.labels[index].text)
        XCTAssertNotEqual(viewController.labels[index].text, "1")
        viewController.updateLabel(at: index, text: "1")
        XCTAssertEqual(viewController.labels[index].text, "1")
    }

    func testIncrementButtonTap() {
        XCTAssertEqual(presenter.incrementButtonTapCount, 0)
        viewController.incrementButton.sendActions(for: .touchUpInside)
        XCTAssertEqual(presenter.incrementButtonTapCount, 1)
    }
}
```

MVVM


ViewModel
Fetch / Update
Data Binding
Model
View

RxSwift



README.md	Move word inside paranthesis	3 months ago
RxBl podspec	Release 3.6.1	2 months ago
Co	ease 3.6.1	2 months ago
RxSwift.podspec	Release 3.6.1	2 months ago
RxTest.podspec	Release 3.6.1	2 months ago

README.md

RxSwift: ReactiveX for Swift

build passing | platforms iOS | macOS | tvOS | watchOS | Linux | pod v4.0.0-alpha.1 | Carthage compatible | Swift Package Manager compatible

Rx is a generic abstraction of computation expressed through `Observable<Element>` interface.

This is a Swift version of Rx.

It tries to port as many concepts from the original version as possible, but some concepts were adapted for more pleasant and performant integration with iOS/macOS environment.

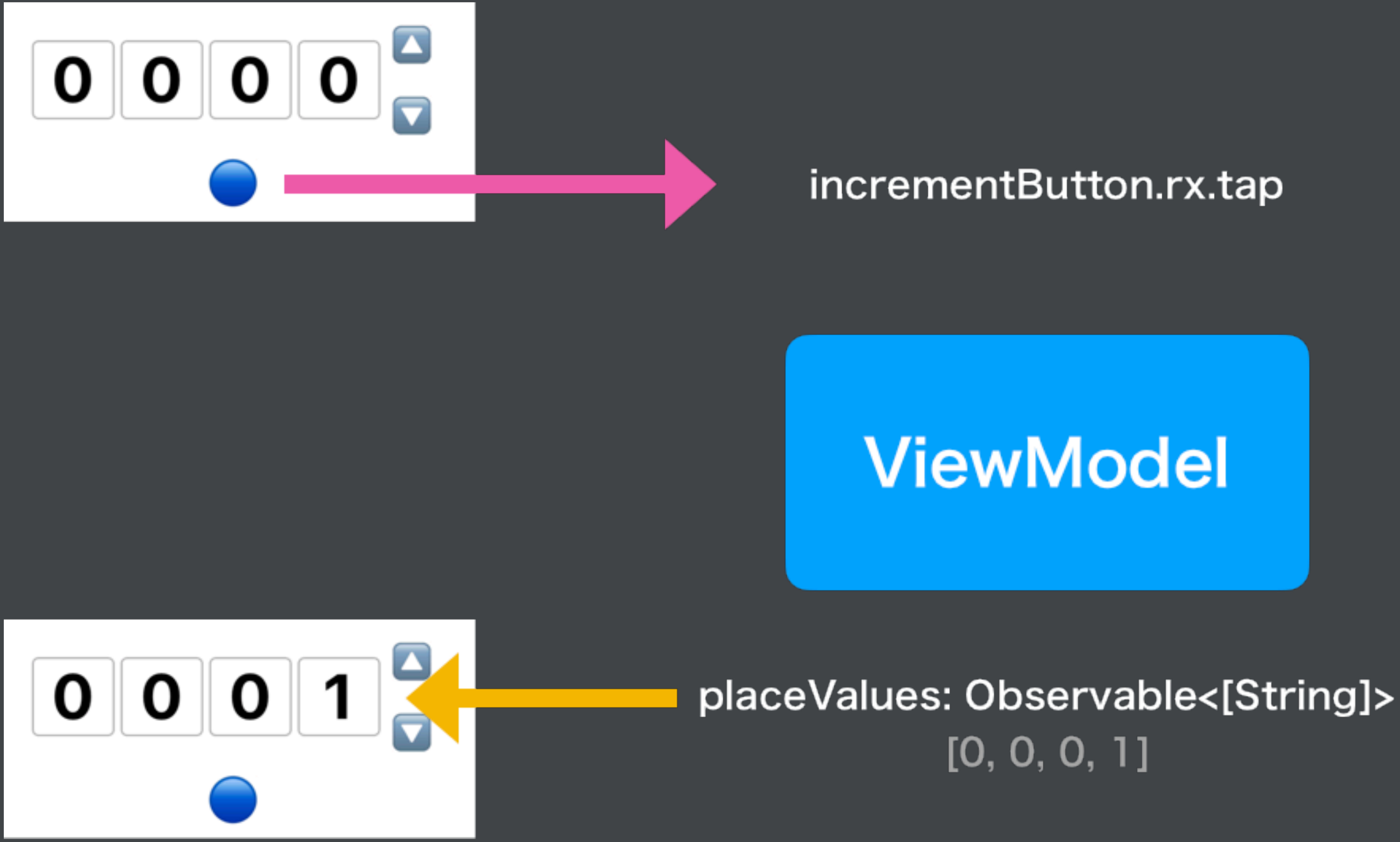
Cross platform documentation can be found on ReactiveX.io.

Like the original Rx, its intention is to enable easy composition of asynchronous operations and event/data streams.

KVO observing, async operations and streams are all unified under abstraction of sequence. This is the reason why Rx is so simple, elegant and powerful.

```swift
final class MVVMViewController: UIViewController {
    @IBOutlet private(set) var labels: [UILabel]!
    @IBOutlet private(set) weak var incrementButton: UIButton!
    @IBOutlet private(set) weak var upButton: UIButton!
    @IBOutlet private(set) weak var downButton: UIButton!

    private let disposeBag = DisposeBag()
    private lazy var viewModel: CounterViewModel = {
        .init(numberOfDigits: self.labels.count,
              incrementButtonTap: self.incrementButton.rx.tap.asObservable(),
              upButtonTap: self.upButton.rx.tap.asObservable(),
              downButtonTap: self.downButton.rx.tap.asObservable())
    }()

    override func viewDidLoad() {
        super.viewDidLoad()

        viewModel.placeValues
            .bind(to: Binder(self) { me, values in
                values.enumerated().forEach { me.labels[$0].text = $1 }
            })
            .disposed(by: disposeBag)
    }
}
```

```swift
final class CounterViewModel {
    let placeValues: Observable<[String]>
    private let disposeBag = DisposeBag()
    ... // other properties

    init(numberOfDigits: Int,
         incrementButtonTap: Observable<Void>,
         upButtonTap: Observable<Void>,
         downButtonTap: Observable<Void>) {
        ... // other implementations

        incrementButtonTap
            .subscribe(onNext: { ... // increment implementations })
            .disposed(by: disposeBag)

        upButtonTap
            .subscribe(onNext: { ... // up implementations })
            .disposed(by: disposeBag)

        downButtonTap
            .subscribe(onNext: { ... // down implementations })
            .disposed(by: disposeBag)
    }
}
```

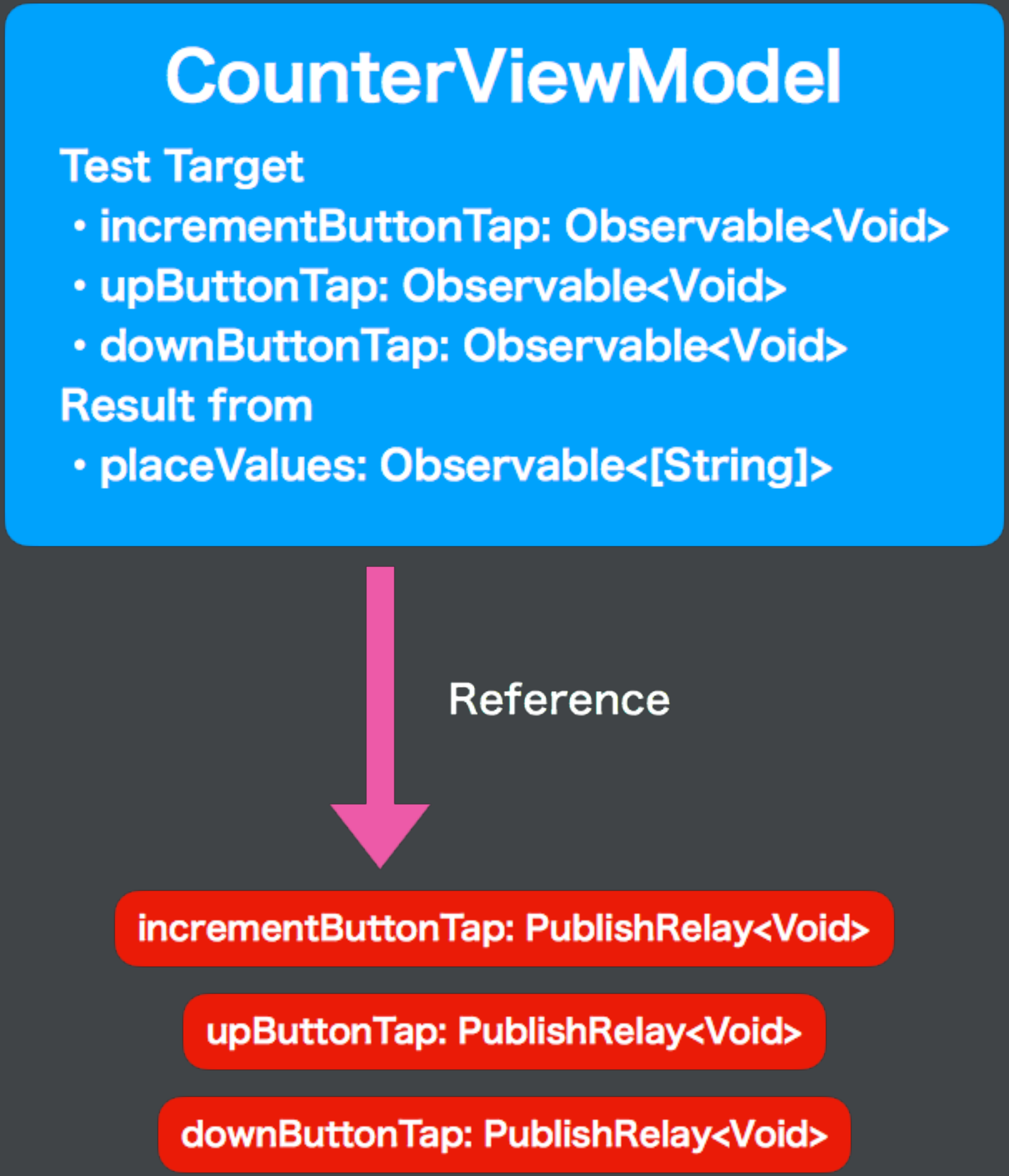

0 0 0 0
incrementButton.rx.tap
ViewModel
0 0 0 1
placeValues: Observable<[String]>
[0, 0, 0, 1]

placeValues: Observable<[String]>
Hot or Cold and so on... 🤔

MVVM Unit Test

ViewModel Unit Test

```swift
final class CounterViewModelTestCase: XCTestCase {
    private var viewModel: CounterViewModel!
    private var incrementButtonTap: PublishRelay<Void>!
    private var upButtonTap: PublishRelay<Void>!
    private var downButtonTap: PublishRelay<Void>!

    private var disposeBag: DisposeBag!
    private var results: BehaviorRelay<[String]>!

    override func setUp() {
        super.setUp()

        let incrementButtonTap = PublishRelay<Void>()
        let upButtonTap = PublishRelay<Void>()
        let downButtonTap = PublishRelay<Void>()
        self.viewModel = CounterViewModel(numberOfDigits: 4,
                                          incrementButtonTap: incrementButtonTap.asObservable(),
                                          upButtonTap: upButtonTap.asObservable(),
                                          downButtonTap: downButtonTap.asObservable())
        self.incrementButtonTap = incrementButtonTap
        self.upButtonTap = upButtonTap
        self.downButtonTap = downButtonTap

        let disposeBag = DisposeBag()
        self.disposeBag = disposeBag
        let results = BehaviorRelay<[String]>(value: [])
        viewModel.placeValues
            .bind(to: results)
            .disposed(by: disposeBag)
        self.results = results
    }

    ... // other implementations
}
```



CounterViewModel

Test Target
- incrementButtonTap: Observable<Void>
- upButtonTap: Observable<Void>
- downButtonTap: Observable<Void>

Result from
- placeValues: Observable<[String]>

incrementButtonTap: PublishRelay<Void>

upButtonTap: PublishRelay<Void>

downButtonTap: PublishRelay<Void>

```swift
final class CounterViewModelTestCase: XCTestCase {
    ... // other implementations

    func testInitialValue() {
        XCTAssertEqual(results.value.count, 4)
        results.value.forEach { value in
            XCTAssertEqual(value, "0")
        }
    }

    func testIncrementButtonTap() {
        let lastValue = results.value
        XCTAssertEqual(lastValue.count, 4)
        incrementButtonTap.accept(())

        XCTAssertNotEqual(lastValue, results.value)
        XCTAssertEqual(results.value.count, 4)
        XCTAssertEqual(results.value[0], "1")
        XCTAssertEqual(results.value[1], "0")
        XCTAssertEqual(results.value[2], "0")
        XCTAssertEqual(results.value[3], "0")
    }
}
```

UIViewController Unit Test

Depends on ... 🤔

```
final class MVVMViewController: UIViewController {
    ... // other properties

    private lazy var viewModel: CounterViewModel = {
        .init(numberOfDigits: self.labels.count,
              incrementButtonTap: self.incrementButton.rx.tap.asObservable(),
              upButtonTap: self.upButton.rx.tap.asObservable(),
              downButtonTap: self.downButton.rx.tap.asObservable())
    }()

    ... //  other implementations
}
```

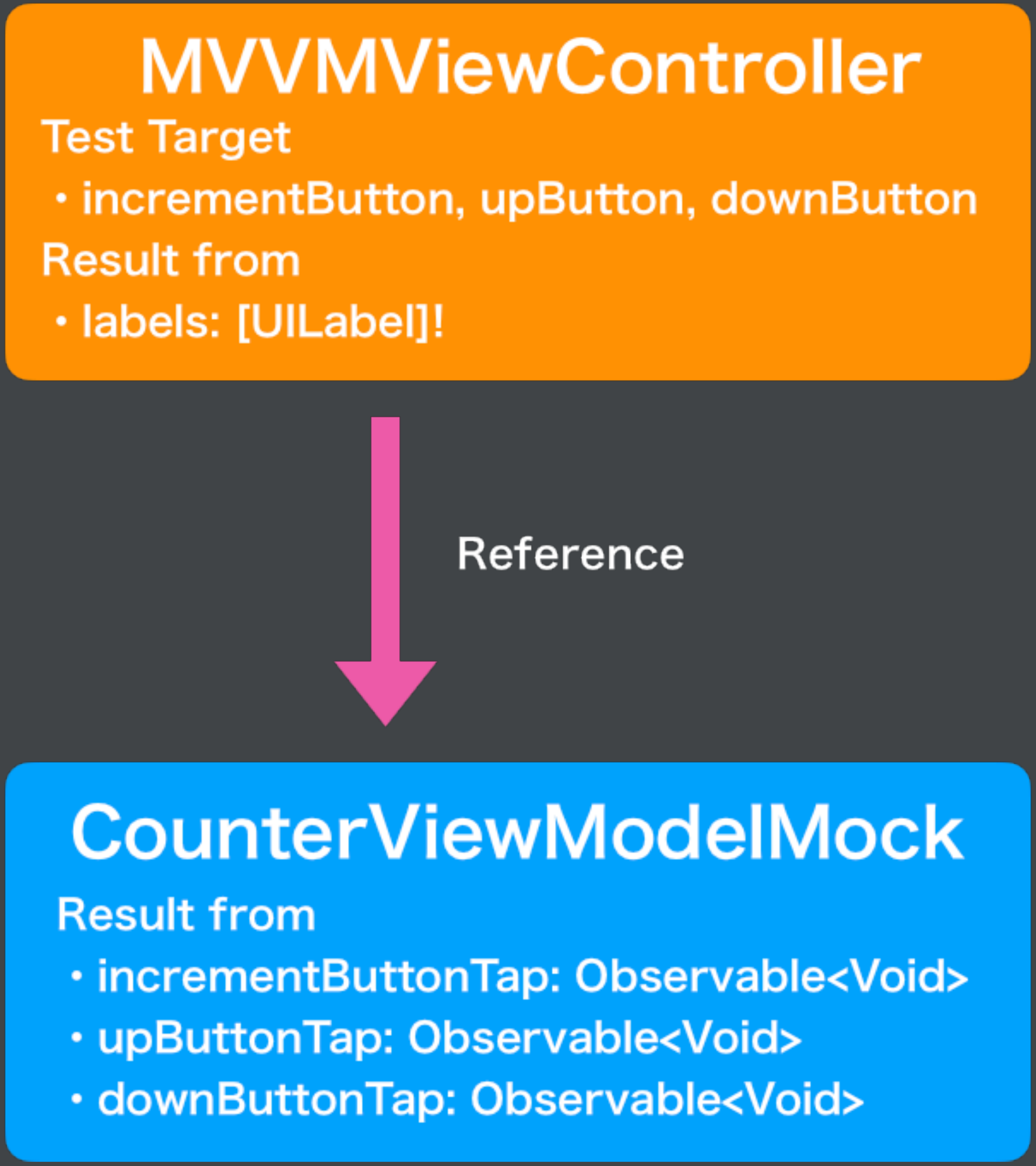
```swift
protocol CounterViewModelType: class {
    var placeValues: Observable<[String]> { get }
    init(numberOfDigits: Int,
         incrementButtonTap: Observable<Void>,
         upButtonTap: Observable<Void>,
         downButtonTap: Observable<Void>)
}

final class CounterViewModel: CounterViewModelType {
    ... // other implementations
}
```

```swift
final class MVVMViewController<ViewModel: CounterViewModelType>: UIViewController {
    ... // other properties

    private lazy var viewModel: ViewModel = {
        .init(numberOfDigits: self.labels.count,
              incrementButtonTap: self.incrementButton.rx.tap.asObservable(),
              upButtonTap: self.upButton.rx.tap.asObservable(),
              downButtonTap: self.downButton.rx.tap.asObservable())
    }()

    init() {
        super.init(nibName: "MVVMViewController", bundle: nil)
    }

    ... // other implementations
}
```

Ready for Testing 👍

```swift
final class CounterViewModelMock: CounterViewModelType {
    let placeValues: Observable<[String]>
    let _placeValues = PublishRelay<[String]>()

    let numberOfDigits: Int
    private(set) var incrementButtonTapCount: Int = 0
    private(set) var upButtonTapCount: Int = 0
    private(set) var downButtonTapCount: Int = 0

    private let disposeBag = DisposeBag()

    init(numberOfDigits: Int,
         incrementButtonTap: Observable<Void>,
         upButtonTap: Observable<Void>,
         downButtonTap: Observable<Void>) {
        self.placeValues = _placeValues.asObservable()
        self.numberOfDigits = numberOfDigits

        incrementButtonTap
            .subscribe(onNext: { [weak self] in self?.incrementButtonTapCount += 1 })
            .disposed(by: disposeBag)

        ... // other implementations
    }
}
```

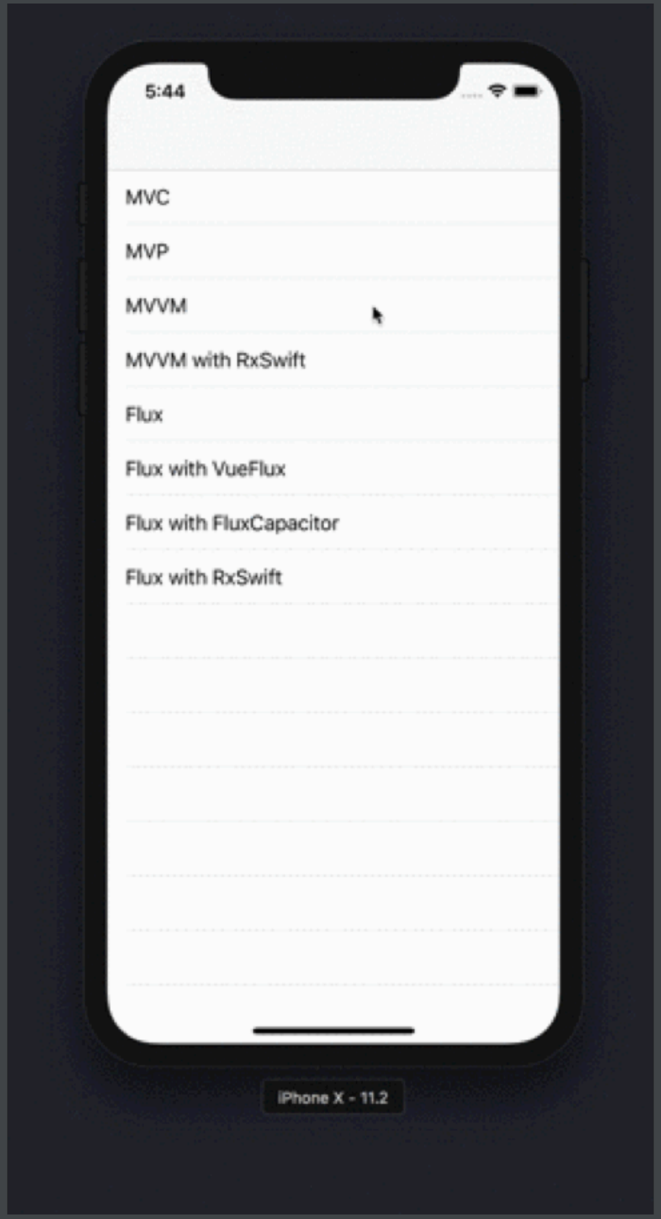
```swift
final class MVVMViewControllerTestCase: XCTestCase {
    private var viewController: MVVMViewController!
    private var viewModel: CounterViewModelMock!
    private var placeValues: PublishRelay<[String]>!

    override func setUp() {
        super.setUp()

        let viewController = MVVMViewController<CounterViewModelMock>()
        _ = viewController.view
        self.viewController = viewController
        self.viewModel = viewController.viewModel
        self.placeValues = viewController.viewModel._placeValues
    }

    ... // other implementations
}
```


MVVMViewController
Test Target
・incrementButton, upButton, downButton
Result from
・labels: [UILabel]!
Reference
CounterViewModelMock
Result from
・incrementButtonTap: Observable<Void>
・upButtonTap: Observable<Void>
・downButtonTap: Observable<Void>

```swift
final class MVVMViewControllerTestCase: XCTestCase {
    ... // other implementations

    func testNumberOfPlaceValues() {
        XCTAssertEqual(viewController.labels.count, viewModel.numberOfDigits)
    }

    func testUpdateLabelAtIndex0To1() {
        let index: Int = 0
        XCTAssertNotNil(viewController.labels[index].text)
        XCTAssertNotEqual(viewController.labels[index].text, "1")
        placeValues.accept(["1", "0", "0", "0"])
        XCTAssertEqual(viewController.labels[index].text, "1")
    }

    func testIncrementButtonTap() {
        XCTAssertEqual(viewModel.incrementButtonTapCount, 0)
        viewController.incrementButton.sendActions(for: .touchUpInside)
        XCTAssertEqual(viewModel.incrementButtonTapCount, 1)
    }
}
```

MVVM without Reactive Frameworks

https://github.com/marty-suzuki/SimplestCounterSample



MVC
MVP
MVVM
MVVM with RxSwift
Flux
Flux with VueFlux
Flux with FluxCapacitor
Flux with RxSwift

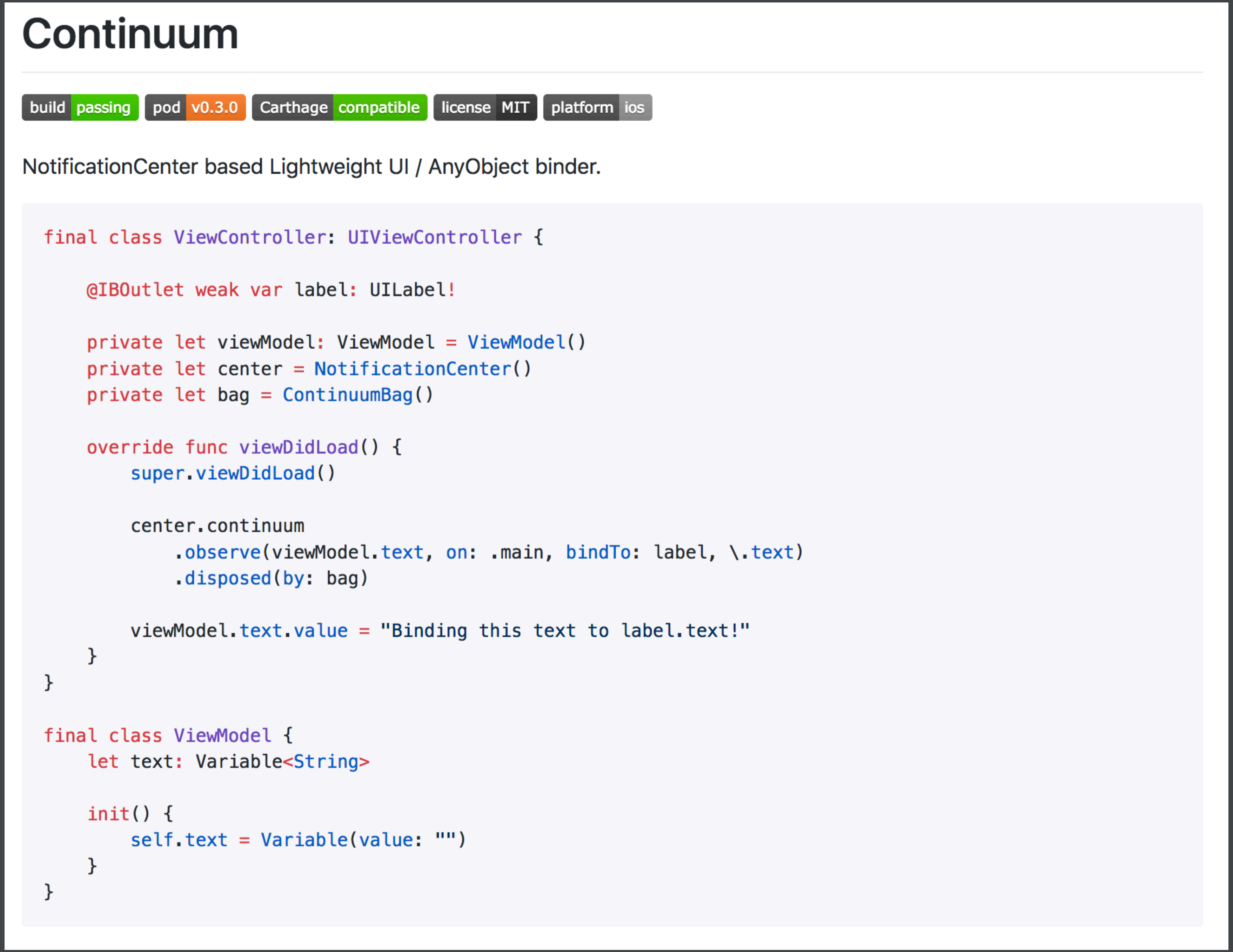
```swift
final class MVVMSampleViewController: UIViewController {
    @IBOutlet private weak var incrementButton: UIButton!
    @IBOutlet private weak var decrementButton: UIButton!
    @IBOutlet private weak var countLabel: UILabel!

    private let viewModel = CountViewModel()

    override func viewDidLoad() {
        super.viewDidLoad()

        incrementButton.addTarget(viewModel,
                                  action: #selector(CountViewModel.increment),
                                  for: .touchUpInside)
        decrementButton.addTarget(viewModel,
                                  action: #selector(CountViewModel.decrement),
                                  for: .touchUpInside)

        viewModel.observe(keyPath: \.count, bindTo: countLabel, \.text)
        ... // other implementations
    }
}
```

```swift
final class CountViewModel {
    private enum Names {
        static let countChanged = Notification.Name(rawValue: "CountViewModel.countChanged")
        ... // other static properties
    }

    ... // other properties

    private var observers: [NSObjectProtocol] = []
    private let center: NotificationCenter

    init(center: NotificationCenter = .init()) {
        self.center = center
        ... // other implementations
    }

    ... // other implementations
}
```

```swift
final class CountViewModel {
    ... // other properties

    private(set) var count: String = "" {
        didSet { center.post(name: Names.countChanged, object: nil) }
    }

    private var _count: Int = 0 {
        didSet {
            count = String(_count)
            ... // other implementations
        }
    }

    ... // other implementations

    @objc func increment() {
        _count += 1
    }

    @objc func decrement() {
        _count -= 1
    }
}
```

```swift
final class CountViewModel {
    ... // other implementations

    func observe<Value1, Target: AnyObject, Value2>(keyPath keyPath1: KeyPath<CountViewModel, Value1>,
                                                     bindTo target: Target,
                                                     _ keyPath2: ReferenceWritableKeyPath<Target, Value2>) {
        let name: Notification.Name
        switch keyPath1 {
        case \CountViewModel.count: name = Names.countChanged
        ... // other cases
        }

        let handler: () -> () = { [weak self, weak target] in
            guard let me = self, let target = target, let value = me[keyPath: keyPath1] as? Value2 else { return }
            target[keyPath: keyPath2] = value
        }

        handler()
        observers.append(center.addObserver(forName: name, object: nil, queue: .main) { _ in handler() })
    }
}
```

https://github.com/marty-suzuki/Continuum



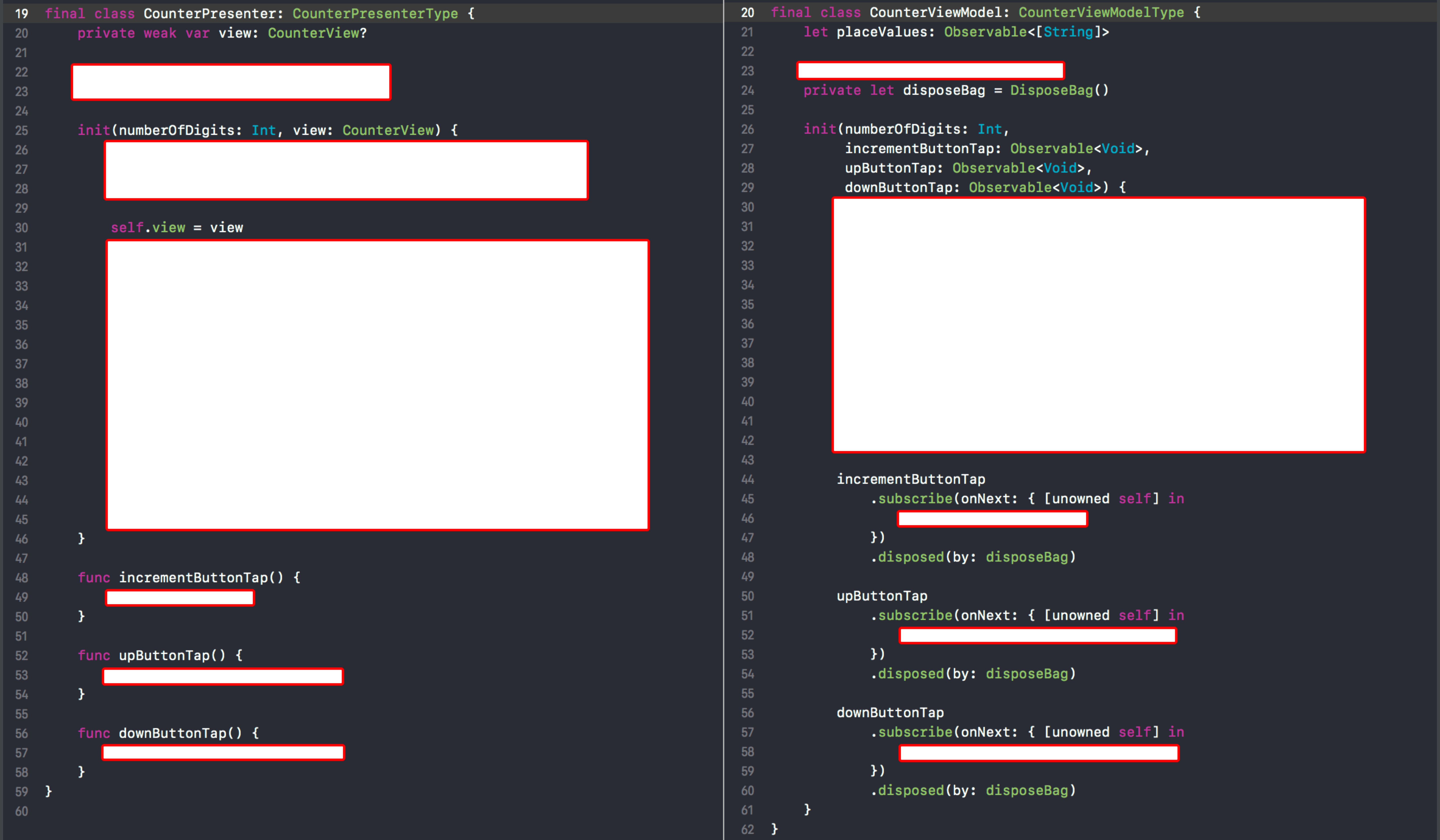
Continuum

build passing | pod v0.3.0 | Carthage compatible | license MIT | platform ios

NotificationCenter based Lightweight UI / AnyObject binder.

```swift
final class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    private let viewModel: ViewModel = ViewModel()
    private let center = NotificationCenter()
    private let bag = ContinuumBag()

    override func viewDidLoad() {
        super.viewDidLoad()

        center.continuum
            .observe(viewModel.text, on: .main, bindTo: label, \.text)
            .disposed(by: bag)

        viewModel.text.value = "Binding this text to label.text!"
    }
}

final class ViewModel {
    let text: Variable<String>

    init() {
        self.text = Variable(value: "")
    }
}
```

Difference between **ViewModel** and **Presenter** in this sample.

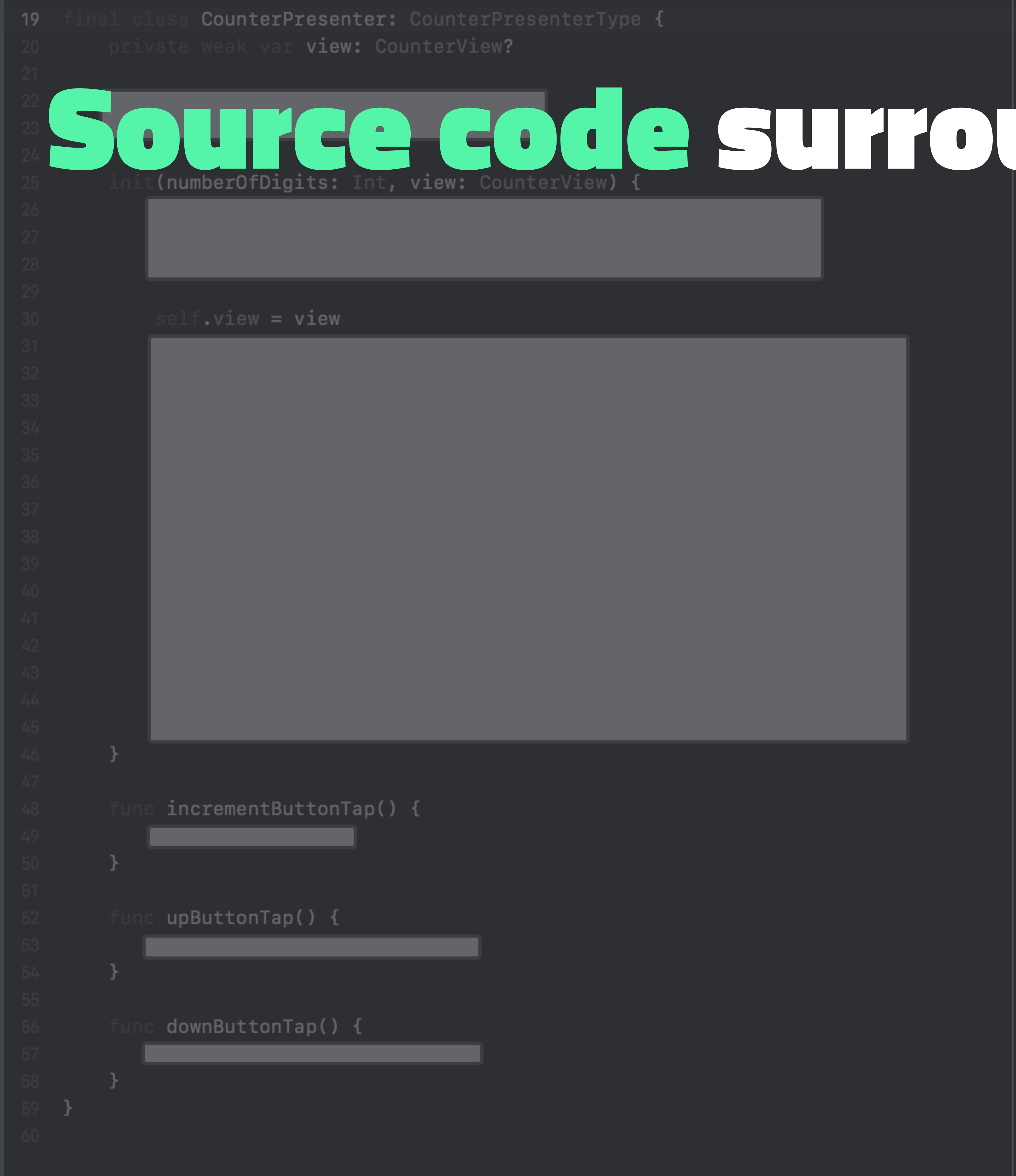


```swift
final class CounterPresenter: CounterPresenterType {
    private weak var view: CounterView?



    init(numberOfDigits: Int, view: CounterView) {



        self.view = view
















    }

    func incrementButtonTap() {

    }

    func upButtonTap() {

    }

    func downButtonTap() {

    }
}
```

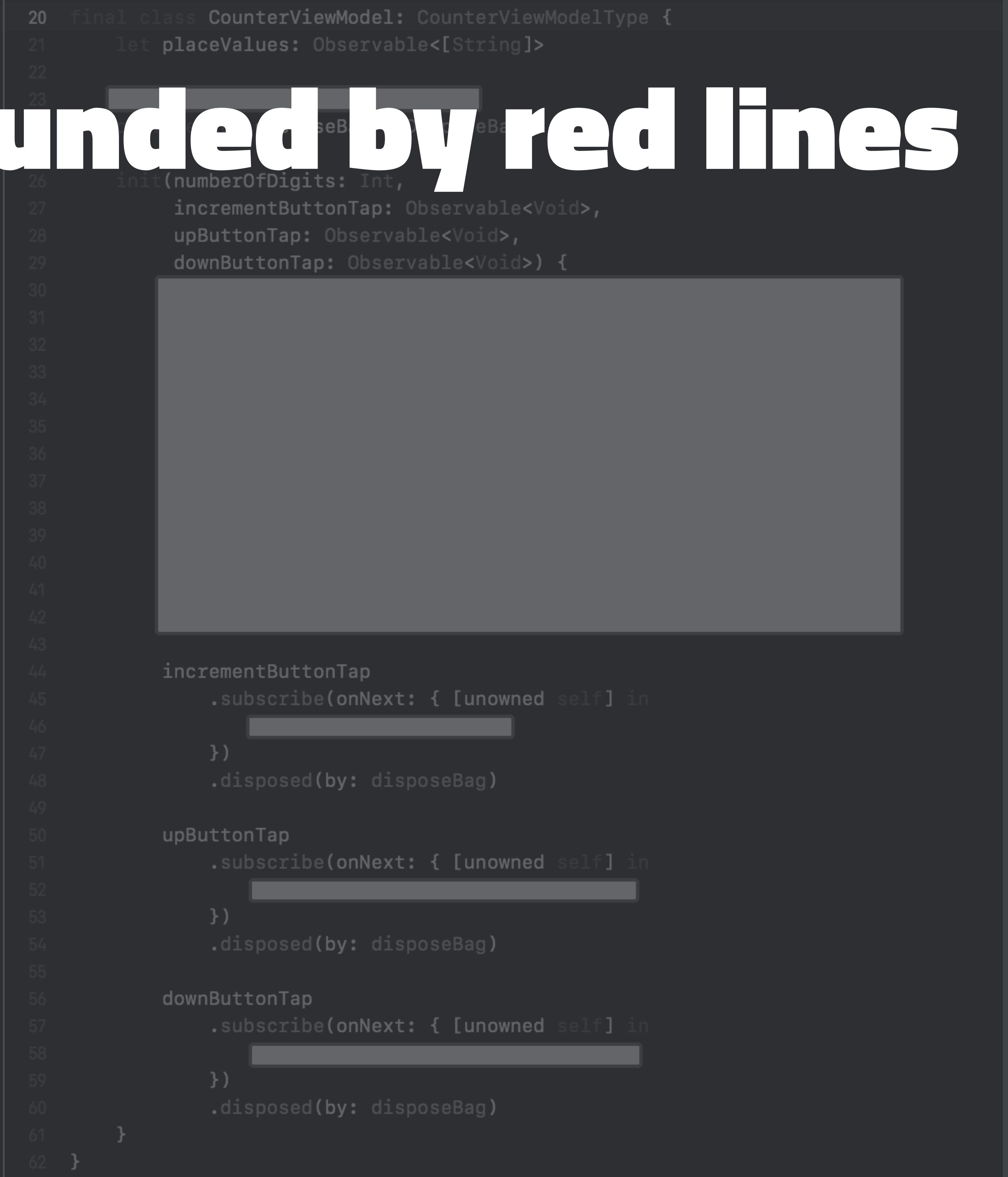
```swift
final class CounterViewModel: CounterViewModelType {
    let placeValues: Observable<[String]>


    private let disposeBag = DisposeBag()

    init(numberOfDigits: Int,
         incrementButtonTap: Observable<Void>,
         upButtonTap: Observable<Void>,
         downButtonTap: Observable<Void>) {














        incrementButtonTap
            .subscribe(onNext: { [unowned self] in

            })
            .disposed(by: disposeBag)

        upButtonTap
            .subscribe(onNext: { [unowned self] in

            })
            .disposed(by: disposeBag)

        downButtonTap
            .subscribe(onNext: { [unowned self] in

            })
            .disposed(by: disposeBag)
    }
}
```

ViewModel Interface and Presenter Interface are different.

Source code surrounded by red lines



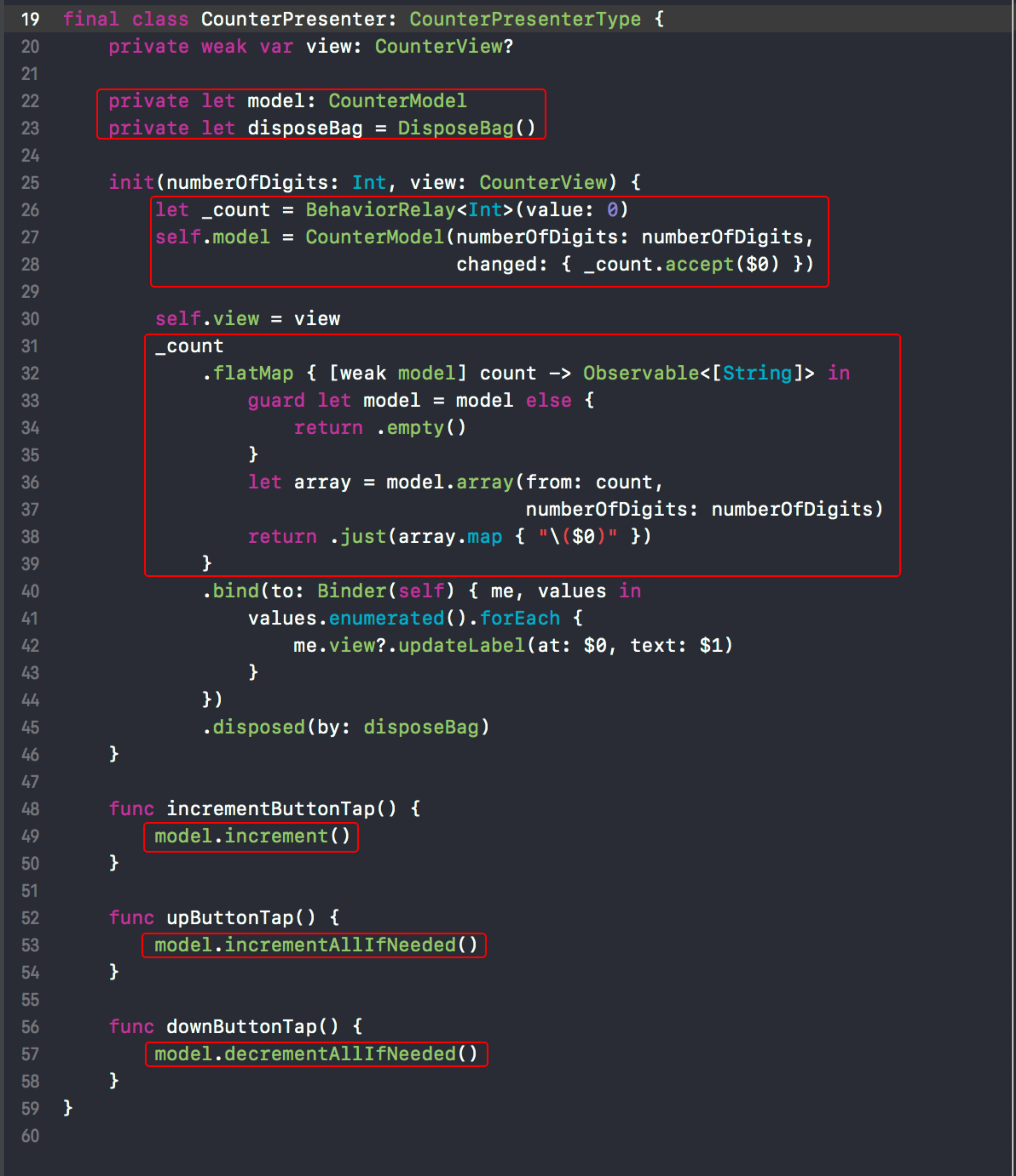
```
final class CounterPresenter: CounterPresenterType {
    private weak var view: CounterView?

    [redacted]

    init(numberOfDigits: Int, view: CounterView) {
        [redacted]

        self.view = view
        [redacted]
    }

    func incrementButtonTap() {
        [redacted]
    }

    func upButtonTap() {
        [redacted]
    }

    func downButtonTap() {
        [redacted]
    }
}
```



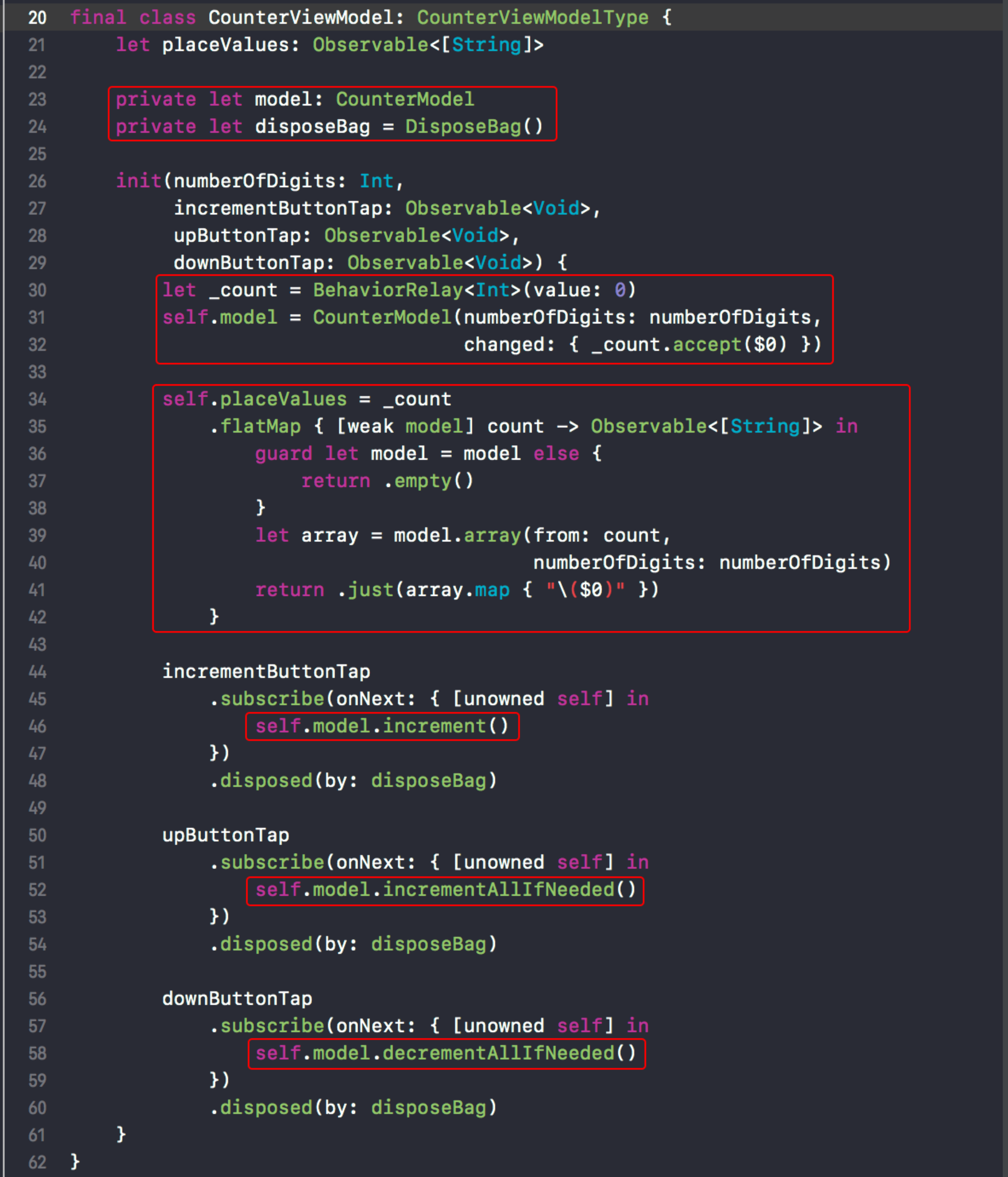
```
final class CounterViewModel: CounterViewModelType {
    let placeValues: Observable<[String]>

    [redacted]

    init(numberOfDigits: Int,
         incrementButtonTap: Observable<Void>,
         upButtonTap: Observable<Void>,
         downButtonTap: Observable<Void>) {
        [redacted]

        incrementButtonTap
            .subscribe(onNext: { [unowned self] in
                [redacted]
            })
            .disposed(by: disposeBag)

        upButtonTap
            .subscribe(onNext: { [unowned self] in
                [redacted]
            })
            .disposed(by: disposeBag)

        downButtonTap
            .subscribe(onNext: { [unowned self] in
                [redacted]
            })
            .disposed(by: disposeBag)
    }
}
```

```swift
final class CounterPresenter: CounterPresenterType {
    private weak var view: CounterView?

    private let model: CounterModel
    private let disposeBag = DisposeBag()

    init(numberOfDigits: Int, view: CounterView) {
        let _count = BehaviorRelay<Int>(value: 0)
        self.model = CounterModel(numberOfDigits: numberOfDigits,
                                  changed: { _count.accept($0) })

        self.view = view
        _count
            .flatMap { [weak model] count -> Observable<[String]> in
                guard let model = model else {
                    return .empty()
                }
                let array = model.array(from: count,
                                        numberOfDigits: numberOfDigits)
                return .just(array.map { "\($0)" })
            }
            .bind(to: Binder(self) { me, values in
                values.enumerated().forEach {
                    me.view?.updateLabel(at: $0, text: $1)
                }
            })
            .disposed(by: disposeBag)
    }

    func incrementButtonTap() {
        model.increment()
    }

    func upButtonTap() {
        model.incrementAllIfNeeded()
    }

    func downButtonTap() {
        model.decrementAllIfNeeded()
    }
}
```

```swift
final class CounterViewModel: CounterViewModelType {
    let placeValues: Observable<[String]>

    private let model: CounterModel
    private let disposeBag = DisposeBag()

    init(numberOfDigits: Int,
         incrementButtonTap: Observable<Void>,
         upButtonTap: Observable<Void>,
         downButtonTap: Observable<Void>) {
        let _count = BehaviorRelay<Int>(value: 0)
        self.model = CounterModel(numberOfDigits: numberOfDigits,
                                  changed: { _count.accept($0) })

        self.placeValues = _count
            .flatMap { [weak model] count -> Observable<[String]> in
                guard let model = model else {
                    return .empty()
                }
                let array = model.array(from: count,
                                        numberOfDigits: numberOfDigits)
                return .just(array.map { "\($0)" })
            }

        incrementButtonTap
            .subscribe(onNext: { [unowned self] in
                self.model.increment()
            })
            .disposed(by: disposeBag)

        upButtonTap
            .subscribe(onNext: { [unowned self] in
                self.model.incrementAllIfNeeded()
            })
            .disposed(by: disposeBag)

        downButtonTap
            .subscribe(onNext: { [unowned self] in
                self.model.decrementAllIfNeeded()
            })
            .disposed(by: disposeBag)
    }
}
```



```swift
final class CounterPresenter: CounterPresenterType {
    private weak var view: CounterView?

    private let model: CounterModel
    private let disposeBag = DisposeBag()

    init(numberOfDigits: Int, view: CounterView) {
        let _count = BehaviorRelay<Int>(value: 0)
        self.model = CounterModel(numberOfDigits: numberOfDigits,
                                  changed: { _count.accept($0) })

        self.view = view
        _count
            .flatMap { [weak model] count -> Observable<[String]> in
                guard let model = model else {
                    return .empty()
                }
                let array = model.array(from: count,
                                        numberOfDigits: numberOfDigits)
                return .just(array.map { "\($0)" })
            }
            .bind(to: Binder(self) { me, values in
                values.enumerated().forEach {
                    me.view?.updateLabel(at: $0, text: $1)
                }
            })
            .disposed(by: disposeBag)
    }

    func incrementButtonTap() {
        model.increment()
    }

    func upButtonTap() {
        model.incrementAllIfNeeded()
    }

    func downButtonTap() {
        model.decrementAllIfNeeded()
    }
}

```



```swift
final class CounterViewModel: CounterViewModelType {
    let placeValues: Observable<[String]>

    private let model: CounterModel
    private let disposeBag = DisposeBag()

    init(numberOfDigits: Int,
         incrementButtonTap: Observable<Void>,
         upButtonTap: Observable<Void>,
         downButtonTap: Observable<Void>) {
        let _count = BehaviorRelay<Int>(value: 0)
        self.model = CounterModel(numberOfDigits: numberOfDigits,
                                  changed: { _count.accept($0) })

        self.placeValues = _count
            .flatMap { [weak model] count -> Observable<[String]> in
                guard let model = model else {
                    return .empty()
                }
                let array = model.array(from: count,
                                        numberOfDigits: numberOfDigits)
                return .just(array.map { "\($0)" })
            }

        incrementButtonTap
            .subscribe(onNext: { [unowned self] in
                self.model.increment()
            })
            .disposed(by: disposeBag)

        upButtonTap
            .subscribe(onNext: { [unowned self] in
                self.model.incrementAllIfNeeded()
            })
            .disposed(by: disposeBag)

        downButtonTap
            .subscribe(onNext: { [unowned self] in
                self.model.decrementAllIfNeeded()
            })
            .disposed(by: disposeBag)
    }
}
```

ViewModel Logic
and
Presenter Logic
can be almost same.

1. Interfaces are **different**.

- A **Presenter** has a reference of **View**, and calls methods of **View**.
- A **ViewModel** publishes changes with callbacks and so on, and a **View** observes them.

2. Logics can be **almost same**.

https://github.com/marty-suzuki/DiffMVPAndMVVM

marty-suzuki / DiffMVPAndMVVM Private

MVP and MVVM implementations and test code sample.

marty-suzuki update README — Latest commit e7d5eb1 22 minutes ago

Carthage/Checkouts	use RxSwift in MVVM	a month ago
DiffMVPAndMVVM.xcodeproj	add CounterModelTestCase	15 hours ago
DiffMVPAndMVVM	add CounterModelTestCase	15 hours ago
DiffMVPAndMVVMTests	add test	3 hours ago
Images	add image	24 minutes ago
.gitignore	add Project	a month ago
.gitmodules	add Project	a month ago
Cartfile	add Project	a month ago
Cartfile.resolved	add Project	a month ago
LICENSE	Initial commit	a month ago
README.md	update README	2 minutes ago

Thank you for listening 😊