상속에서 프로토콜로

프로토콜 활용법

Kofktu [코프]
https://github.com/Kofktu

상속에서 프로토콜로

(younatics/Dismissable)

상속에서 프로토콜로
(younatics/Dismissable)

프로토콜 활용
(Kofktu/KUIPopOver)

상속에서 프로토콜로

(younatics/Dismissable)

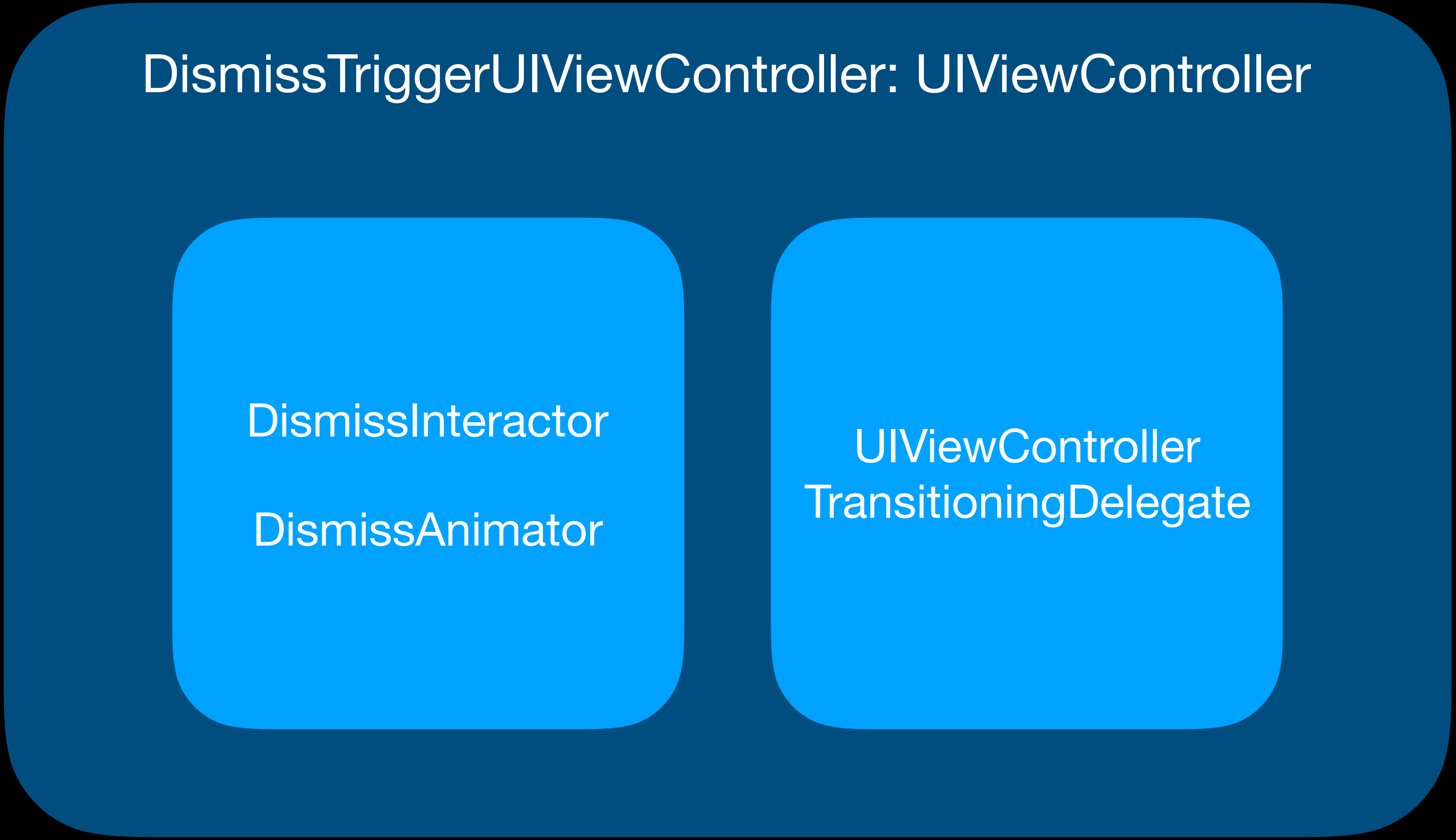
younatics/Dismissable

```
DismissTriggerUIViewController: UIViewController

DismissableUIViewController: UIViewController
```

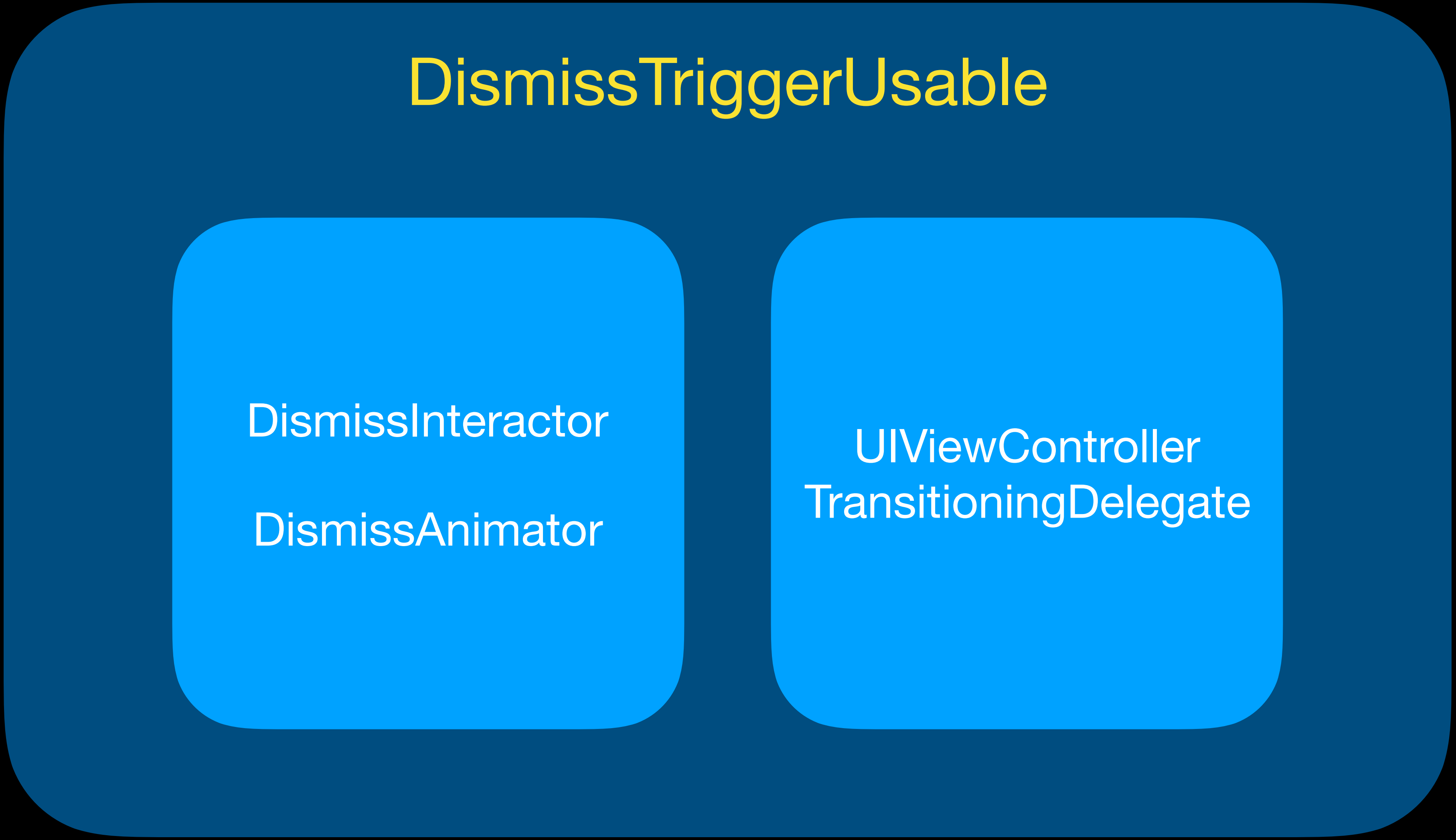
younatics/Dismissable

DismissTriggerUIViewController: UIViewController

younatics/Dismissable


DismissTriggerUIViewController: UIViewController
DismissInteractor
DismissAnimator
UIViewController TransitioningDelegate

younatics/Dismissable



younatics/Dismissable

```swift
public typealias DismissTriggerViewController = (UIViewController & DismissTriggerUsable)

public protocol DismissTriggerUsable {
    var dismissInteractor: DismissInteractor { get }
    var dismissAnimator: DismissAnimator { get }
}
```

DismissTriggerUsable

younatics/Dismissable

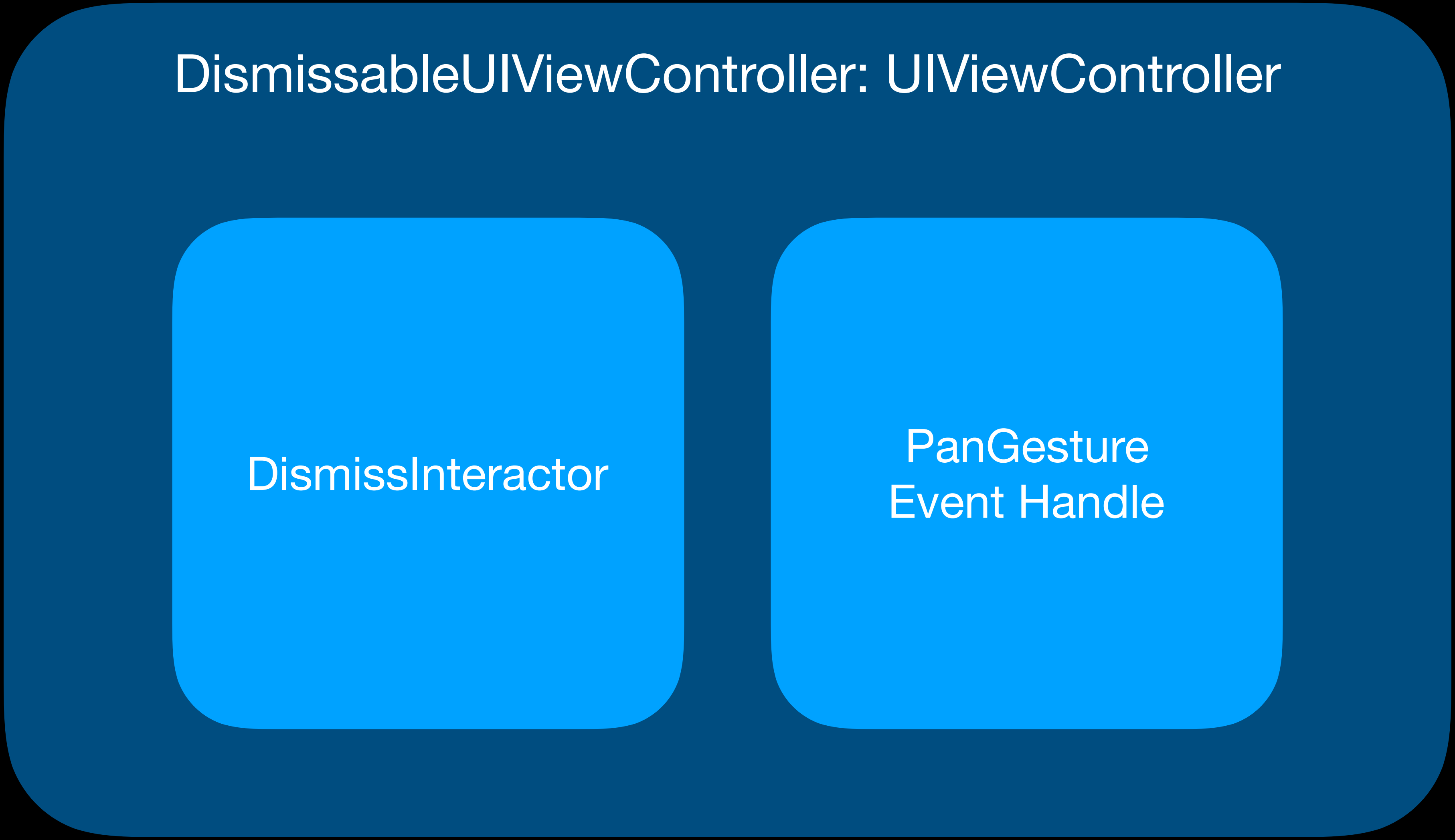
```swift
final class DismissTriggerTransitioningDelegate: NSObject, UIViewControllerTransitioningDelegate {
    private weak var rootViewController: DismissTriggerViewController?

    init(rootViewController: DismissTriggerViewController) {}
    func animationController(forDismissed dismissed: UIViewController)
        -> UIViewControllerAnimatedTransitioning? { return rootViewController?.dismissAnimator }
    func interactionControllerForDismissal(using animator: UIViewControllerAnimatedTransitioning)
        -> UIViewControllerInteractiveTransitioning? { return rootViewController?.dismissInteractor }
}
```

DismissTriggerTransitioningDelegate

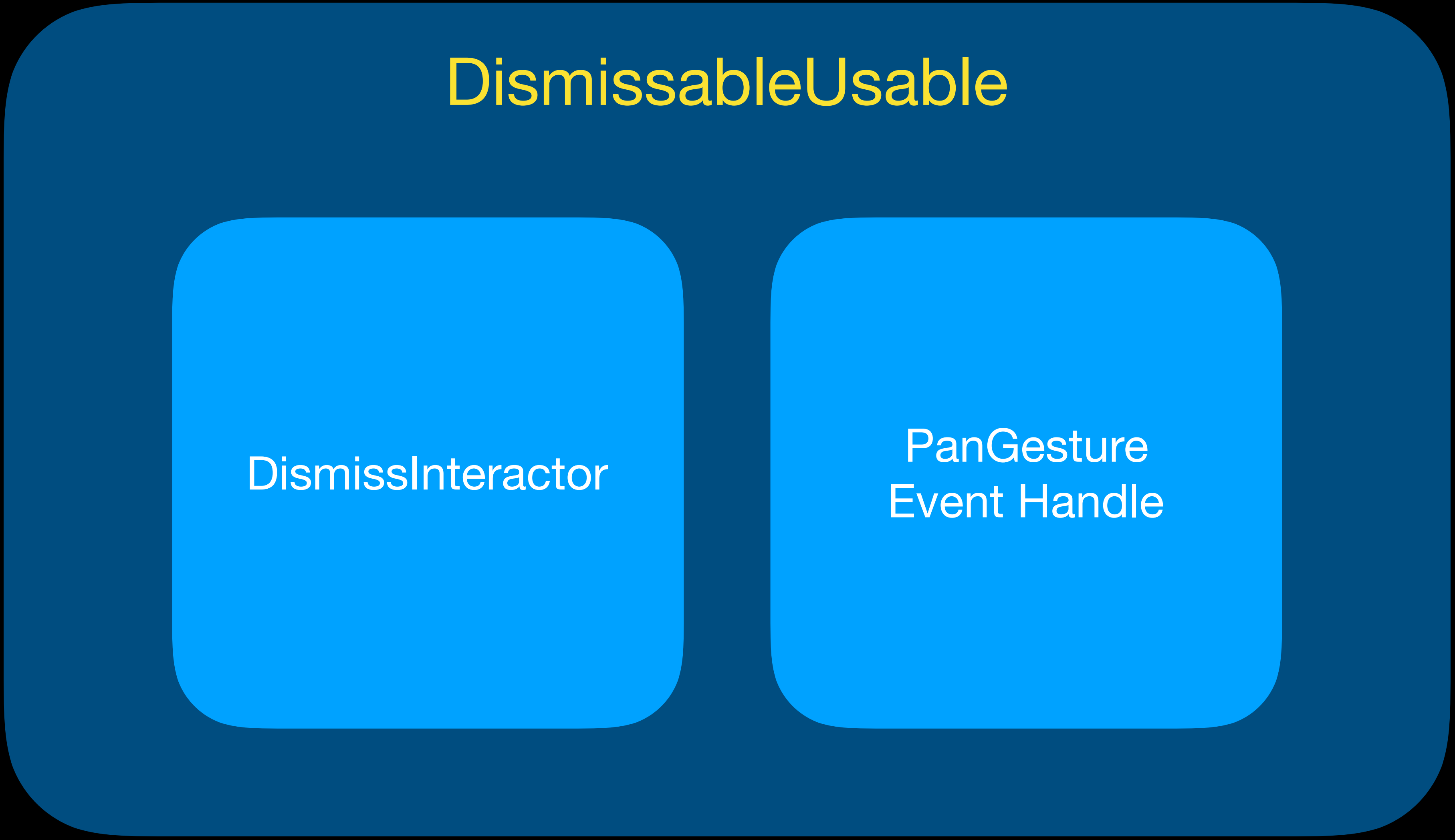
younatics/Dismissable

DismissableUIViewController: UIViewController

younatics/Dismissable



younatics/Dismissable



younatics/Dismissable

```swift
public typealias DismissableViewController = (UIViewController & DismissableUsable)

public protocol DismissableUsable {
    var percentThreshold: CGFloat { get }
    func setup(dismissable: (vc: DismissTriggerViewController,
        dismissInteractor: DismissInteractor))
}
```

DismissableUsable

younatics/Dismissable

```swift
final class DismissableUsableEventDispatcher: NSObject, UIGestureRecognizerDelegate {
    private unowned var rootViewController: DismissableViewController
    init(rootViewController: DismissableViewController) {}
    @objc func onPanGesture(_ gesture: UIPanGestureRecognizer) {}
}
```

DismissableUsableEventDispatcher

저렇게 분류만 하면 되나요?

No!
조금만 더 알아봅시다

Protocol Extension

Protocol Extension

Associated Objects

Protocol Extension

Extension Syntax

Protocol Extension

```swift
public extension DismissableUsable where Self: UIViewController {

    mutating func setup(dismissable: (vc: DismissTriggerViewController,
                       dismissInteractor: DismissInteractor)) {
        let dismissTriggerTransitioning = DismissTriggerTransitioningDelegate(rootViewController:
            dismissable.vc)
        self.transitioningDelegate = dismissTriggerTransitioning
        self.dismissableTriggerTransitioning = dismissTriggerTransitioning
        self.dismissableInteractor = dismissable.dismissInteractor
        self.eventDispatcher = DismissableUsableEventDispatcher(rootViewController: self)
    }

}
```

DismissableUsable 을 준수한 UIViewController에만
기능을 제공해주고 싶을때

Protocol Extension

Default Value

Protocol Extension

```swift
public typealias DismissableViewController = (UIViewController & DismissableUsable)

public protocol DismissableUsable {
    var percentThreshold: CGFloat { get }
    func setup(dismissable: (vc: DismissTriggerViewController,
        dismissInteractor: DismissInteractor))
}
```

percentThreshold의 기본값을 정해주고 싶다면!?

Protocol Extension

```swift
public typealias DismissableViewController = (UIViewController & DismissableUsable)

public protocol DismissableUsable {
    var percentThreshold: CGFloat { get }
    func setup(dismissable: (vc: DismissTriggerViewController,
        dismissInteractor: DismissInteractor))
}

public extension DismissableUsable {
    var percentThreshold: CGFloat { 0.3 }
}
```

percentThreshold의 기본값을 0.3 으로 지정

다른 값을 사용하고 싶다면!? 해당 객체에서 재정의

Associated Objects

Associated Objects

objc_setAssociatedObject

objc_getAssociatedObject

objc_removeAssociatedObjects

Associated Objects는 런타임시 사용자 속성이나 메소드들을

서브클래스를 만들지 않고 추가/제거 가능

Associated Objects

```
objc_setAssociatedObject(object: Any, key: UnsafeRawPointer, value: Any?, policy: objc_AssociationPolicy)
objc_getAssociatedObject(object: Any, key: UnsafeRawPointer)
```

객체 인스턴스 변수를 기존 클래스에 추가할 때 사용

Associated Objects

```
objc_setAssociatedObject(object: Any, key: UnsafeRawPointer, value: Any?, policy: objc_AssociationPolicy)
objc_getAssociatedObject(object: Any, key: UnsafeRawPointer)
```

객체 인스턴스 변수를 기존 클래스에 추가할 때 사용

클래스 수정/재정의 없이 임의의 변수를 추가하고 필요할 때 사용

Associated Objects

```
objc_setAssociatedObject(object: Any, key: UnsafeRawPointer, value: Any?, policy: objc_AssociationPolicy)
objc_getAssociatedObject(object: Any, key: UnsafeRawPointer)
```

객체 인스턴스 변수를 기존 클래스에 추가할 때 사용

클래스 수정/재정의 없이 임의의 변수를 추가하고 필요할 때 사용

Association는 **Key** 기반으로 동작

Associated Objects

```
objc_setAssociatedObject(object: Any, key: UnsafeRawPointer, value: Any?, policy: objc_AssociationPolicy)
objc_getAssociatedObject(object: Any, key: UnsafeRawPointer)
```

객체 인스턴스 변수를 기존 클래스에 추가할 때 사용

클래스 수정/재정의 없이 임의의 변수를 추가하고 필요할 때 사용

Association는 **Key** 기반으로 동작

소스객체가 해제되면 소스 객체에 추가된 연관 객체도 해제

Associated Objects

```
extension UIViewController {

    enum AssociatedKeys {
        static var eventDispatcher = "eventDispatcher"
        static var dismissableTriggerTransitioning = "dismissableTriggerTransitioning"
        static var dismissableInteractor = "dismissableInteractor"
    }

    var eventDispatcher: DismissableUsableEventDispatcher? {
        get { return objc_getAssociatedObject(self, &AssociatedKeys.eventDispatcher) as?
            DismissableUsableEventDispatcher }
        set { objc_setAssociatedObject(self, &AssociatedKeys.eventDispatcher, newValue,
            .OBJC_ASSOCIATION_RETAIN_NONATOMIC) }
    }

    var dismissableTriggerTransitioning: DismissTriggerTransitioningDelegate? {
        get { return objc_getAssociatedObject(self, &AssociatedKeys.dismissableTriggerTransitioning) as?
            DismissTriggerTransitioningDelegate }
        set { objc_setAssociatedObject(self, &AssociatedKeys.dismissableTriggerTransitioning, newValue,
            .OBJC_ASSOCIATION_RETAIN_NONATOMIC) }
    }

    var dismissableInteractor: DismissInteractor? {
        get { return objc_getAssociatedObject(self, &AssociatedKeys.dismissableInteractor) as?
            DismissInteractor }
        set { objc_setAssociatedObject(self, &AssociatedKeys.dismissableInteractor, newValue,
            .OBJC_ASSOCIATION_RETAIN_NONATOMIC) }
    }

}
```

적용해봅시다!

younatics/Dismissable

```swift
class ViewController: DismissTriggerUIViewController {

    func loadDetailViewController() {
        let viewController = DetailViewController()
        viewController.dismissable = (self, dismissInteractor)
        present(viewController, animated: true, completion: nil)
    }

}

class DetailViewController: DismissableUIViewController {}
```

younatics/Dismissable

```
class ViewController: UIViewController, DismissTriggerUsable {

    var dismissInteractor: DismissInteractor = DismissInteractor()
    var dismissAnimator: DismissAnimator = DismissAnimator()

    func loadDetailViewController() {
        var viewController = DetailViewController()
        viewController.setup(dismissable: (self, dismissInteractor))
        present(viewController, animated: true, completion: nil)
    }

}

class DetailViewController: UIViewController, DismissableUsable {}
```

😃😃😃

younatics/Dismissable

```swift
public extension DismissTriggerUsable {
    var dismissInteractor: DismissInteractor {
        return DismissInteractor.shared
    }
    var dismissAnimator: DismissAnimator {
        return DismissAnimator.shared
    }
}
class ViewController: UIViewController, DismissTriggerUsable {

    func loadDetailViewController() {
        var viewController = DetailViewController()
        viewController.setup(dismissable: (self, dismissInteractor))
        present(viewController, animated: true, completion: nil)
    }

}

class DetailViewController: UIViewController, DismissableUsable {}
```

👏👏👏👏👏👏

이거까지 알면 되나요?

네!

실제 PR 했던 내용을 확인해봅시다

잘 이해가 안됩니다만...

그러면 또 다른 예시를 보시죠!

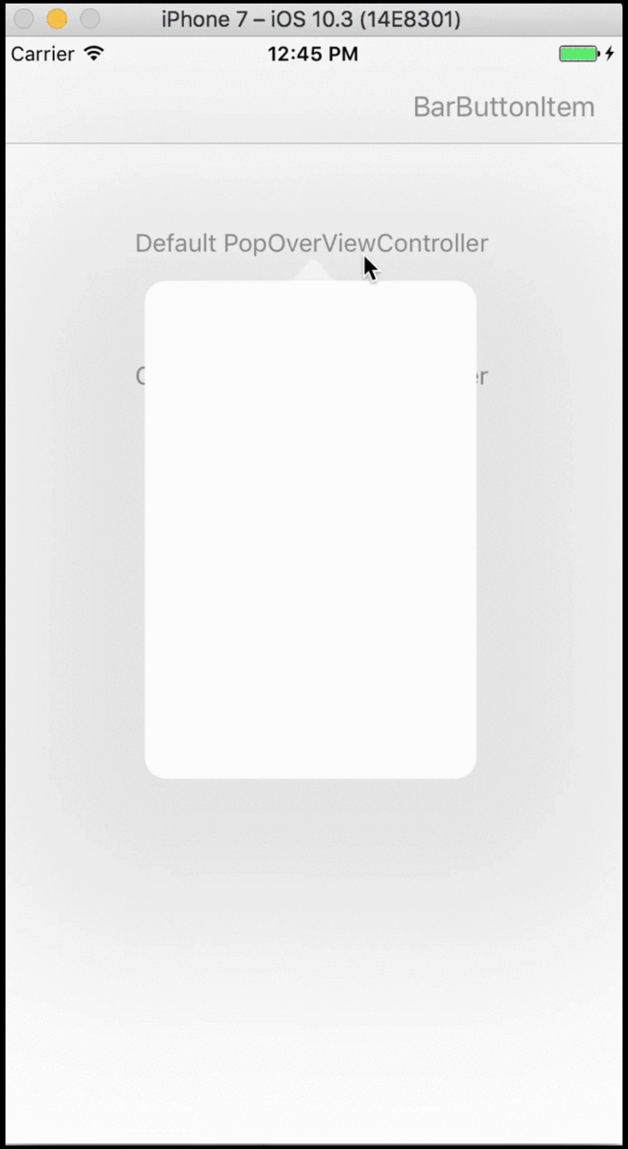
프로토콜 활용

(Kofktu/KUIPopOver)

Kofktu/KUIPopOver

UIPopoverController

UIPopoverPresentationController



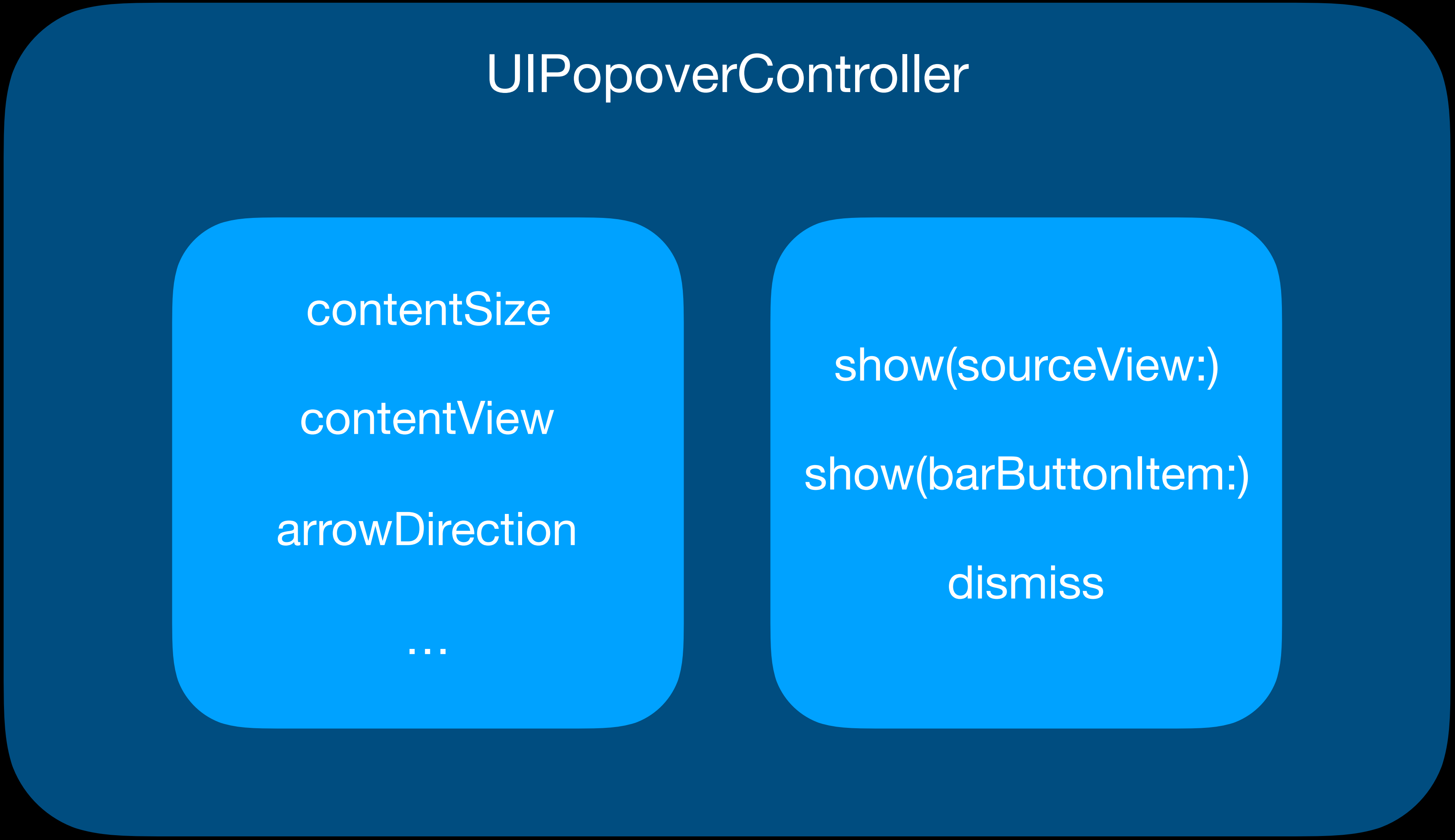
Kofktu/KUIPopOver

```swift
let popOverViewController = DefaultPopOverViewController()
popOverViewController.preferredContentSize = CGSize(width: 200.0, height: 300.0)
popOverViewController.popoverPresentationController?.sourceView = sender

let customView = CustomPopOverView(frame: CGRect(origin: CGPoint(x: 0.0, y: 0.0),
                                                  size: CGSize(width: 200.0, height: 300.0)))
popOverViewController.view.addSubview(customView)
popOverViewController.popoverPresentationController?.sourceRect = sender.bounds
present(popOverViewController, animated: true, completion: nil)
```

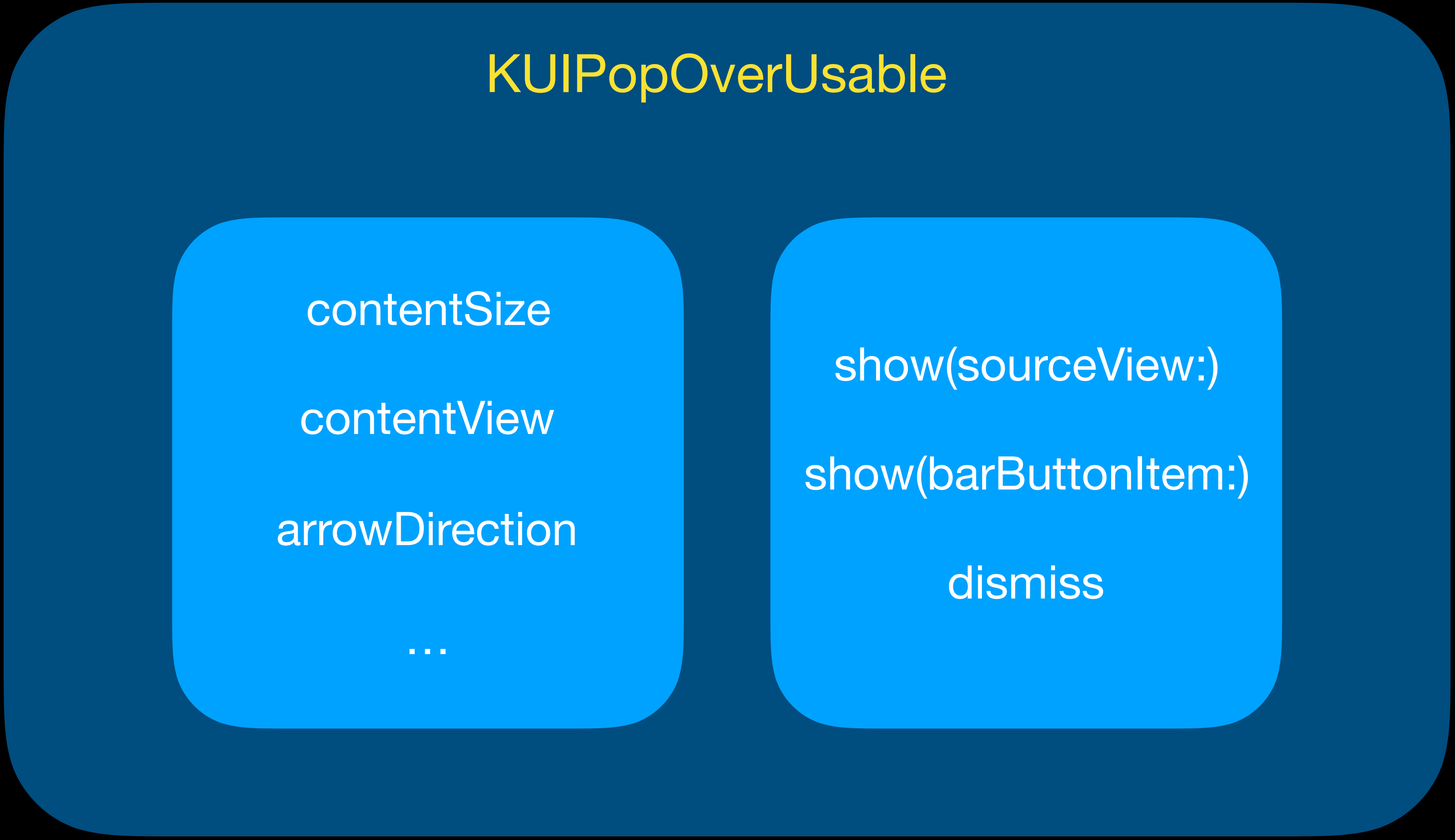
UIPopoverController 를

조금 더 편하게 쓰고 싶어요..ㅠㅠ!

Kofktu/KUIPopOver



Kofktu/KUIPopOver



Kofktu/KUIPopOver

```swift
public protocol KUIPopOverUsable {

    var contentSize: CGSize { get }
    var contentView: UIView { get }
    var popOverBackgroundColor: UIColor? { get }
    var arrowDirection: UIPopoverArrowDirection { get }
}

extension KUIPopOverUsable {

    public var popOverBackgroundColor: UIColor? {
        return nil
    }

    public var arrowDirection: UIPopoverArrowDirection {
        return .any
    }
}
```

KUIPopOverUsable + Default Value

Kofktu/KUIPopOver

```swift
extension KUIPopOverUsable where Self: UIViewController {

    public var contentView: UIView {
        return view
    }

    public func showPopover(sourceView: UIView,
                            sourceRect: CGRect? = nil,
                            shouldDismissOnTap: Bool = true,
                            completion: ShowPopoverCompletion? = nil) {}

    public func showPopoverWithNavigationController(sourceView: UIView,
                                                    sourceRect: CGRect? = nil,
                                                    shouldDismissOnTap: Bool = true,
                                                    completion: ShowPopoverCompletion? = nil) {}

    public func showPopover(barButtonItem: UIBarButtonItem,
                            shouldDismissOnTap: Bool = true,
                            completion: ShowPopoverCompletion? = nil) {}

    public func showPopoverWithNavigationController(barButtonItem: UIBarButtonItem,
                                                    shouldDismissOnTap: Bool = true,
                                                    completion: ShowPopoverCompletion? = nil) {}

    public func dismissPopover(animated: Bool, completion: DismissPopoverCompletion? = nil) {}

}
```

KUIPopOverUsable + Extension Syntax

Kofktu/KUIPopOver

```
extension KUIPopOverUsable where Self: UIView {

    public var contentView: UIView {
        return self
    }

    public var contentSize: CGSize {
        return frame.size
    }

    public func showPopover(sourceView: UIView,
                            sourceRect: CGRect? = nil,
                            shouldDismissOnTap: Bool = true,
                            completion: ShowPopoverCompletion? = nil) {}

    public func showPopover(barButtonItem: UIBarButtonItem,
                            shouldDismissOnTap: Bool = true,
                            completion: ShowPopoverCompletion? = nil) {}

    public func dismissPopover(animated: Bool,
                               completion: DismissPopoverCompletion? = nil) {}

}
```

KUIPopOverUsable + Extension Syntax

Kofktu/KUIPopOver

```swift
fileprivate class KUIPopOverUsableDismissHandlerWrapper {
    typealias DismissHandler = ((Bool, DismissPopoverCompletion?) -> Void)
    var closure: DismissHandler?

    init(_ closure: DismissHandler?) {
        self.closure = closure
    }
}

fileprivate extension UIView {

    struct AssociatedKeys {
        static var onDismissHandler = "onDismissHandler"
    }

    var onDismissHandler: KUIPopOverUsableDismissHandlerWrapper.DismissHandler? {
        get { return (objc_getAssociatedObject(self, &AssociatedKeys.onDismissHandler) as?
            KUIPopOverUsableDismissHandlerWrapper)?.closure }
        set { objc_setAssociatedObject(self, &AssociatedKeys.onDismissHandler,
            KUIPopOverUsableDismissHandlerWrapper(newValue), .OBJC_ASSOCIATION_RETAIN_NONATOMIC) }
    }

}
```

Associated Objects + Wrapper(Closure)

또 적용해봅시다!

Kofktu/KUIPopOver

```swift
let popOverViewController = DefaultPopOverViewController()
popOverViewController.preferredContentSize = CGSize(width: 200.0, height: 300.0)
popOverViewController.popoverPresentationController?.sourceView = sender

let customView = CustomPopOverView(frame: CGRect(origin: CGPoint(x: 0.0, y: 0.0),
                                                 size: CGSize(width: 200.0, height: 300.0)))
popOverViewController.view.addSubview(customView)
popOverViewController.popoverPresentationController?.sourceRect = sender.bounds
present(popOverViewController, animated: true, completion: nil)
```

😭😱😭😱

Kofktu/KUIPopOver

```swift
class PopOverView: UIView, KUIPopOverUsable {

    var contentSize: CGSize {
        return CGSize(width: 300.0, height: 400.0)
    }

    var popOverBackgroundColor: UIColor? {
        return .black
    }

    var arrowDirection: UIPopoverArrowDirection {
        return .up
    }
                         😊😊😊😊😊😊
}

let popOverView = PopOverView()
popOverView.showPopover(sourceView: UIView,
                        sourceRect: CGRect?,
                        shouldDismissOnTap: Bool,
                        completion: ShowPopoverCompletion?)
popOverView.showPopover(barButtonItem: UIBarButtonItem,
                        shouldDismissOnTap: Bool,
                        completion: ShowPopoverCompletion?)
popOverView.dismissPopover(animated: Bool,
                           completion: DismissPopoverCompletion?)
```

Kofktu/KUIPopOver

```swift
class PopOverViewController: UIViewController, KUIPopOverUsable {

    var contentSize: CGSize {
        return CGSize(width: 300.0, height: 400.0)
    }

    var popOverBackgroundColor: UIColor? {
        return .blue
    }
                                                  😊😊😊😊😊😊
}

let viewController = PopOverViewController()
viewController.showPopover(sourceView: UIView,
                           sourceRect: CGRect?,
                           shouldDismissOnTap: Bool,
                           completion: ShowPopoverCompletion?)
viewController.showPopover(barButtonItem: UIBarButtonItem,
                           shouldDismissOnTap: Bool,
                           completion: ShowPopoverCompletion?)
viewController.dismissPopover(animated: Bool,
                              completion: DismissPopoverCompletion?)
```

Summary

Summary

기능별로 잘 분리 한다

Summary

기능별로 잘 분리 한다

Protocol Extension

- Extension Syntax
- Default Value

Summary

기능별로 잘 분리 한다

Protocol Extension

Associated Objects

필요한 임의의 변수에 대해서만 추가

악마의 열매이므로 적당히 쓴다!