VIPER architecture

~ MVVM with a little more

View

Interactor

Presenter

Entity

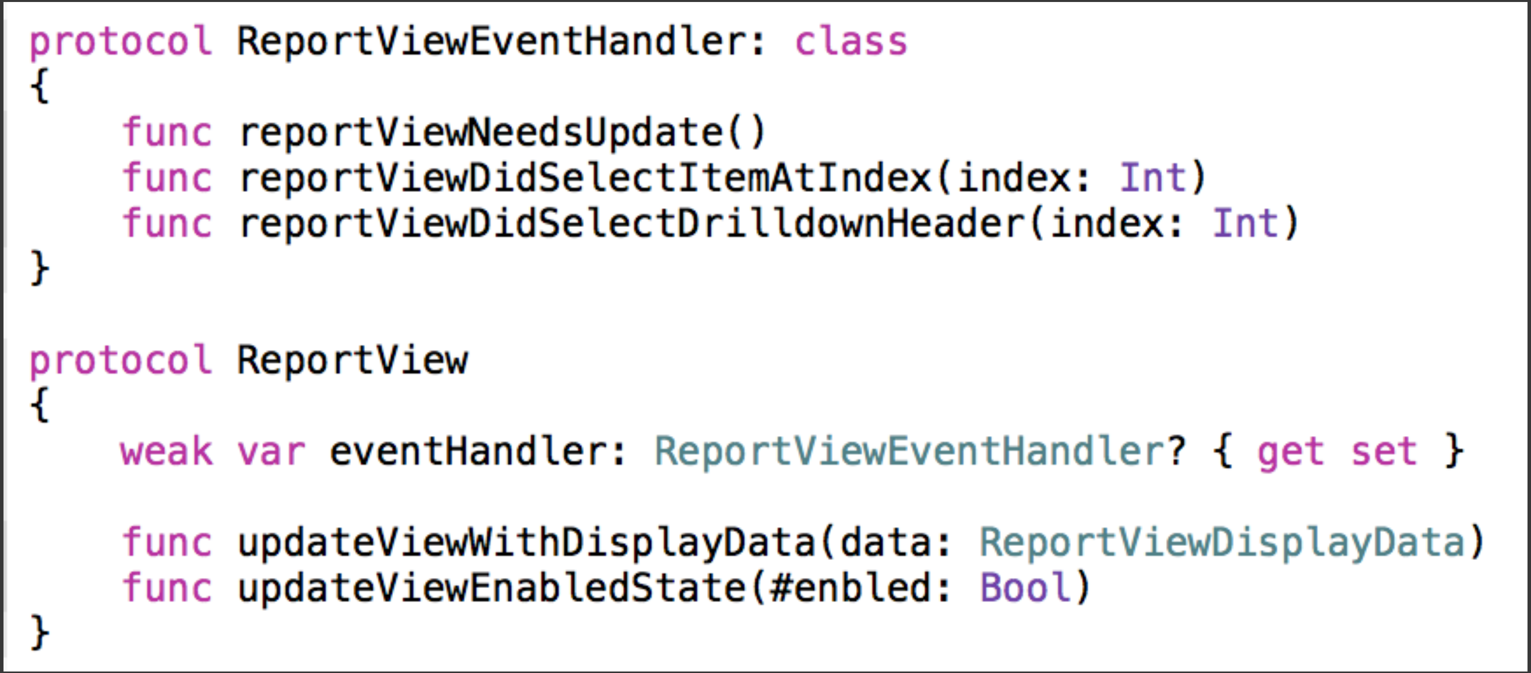
Router

View

describes an interface (protocol) of what a certain view can do / what events can it emit

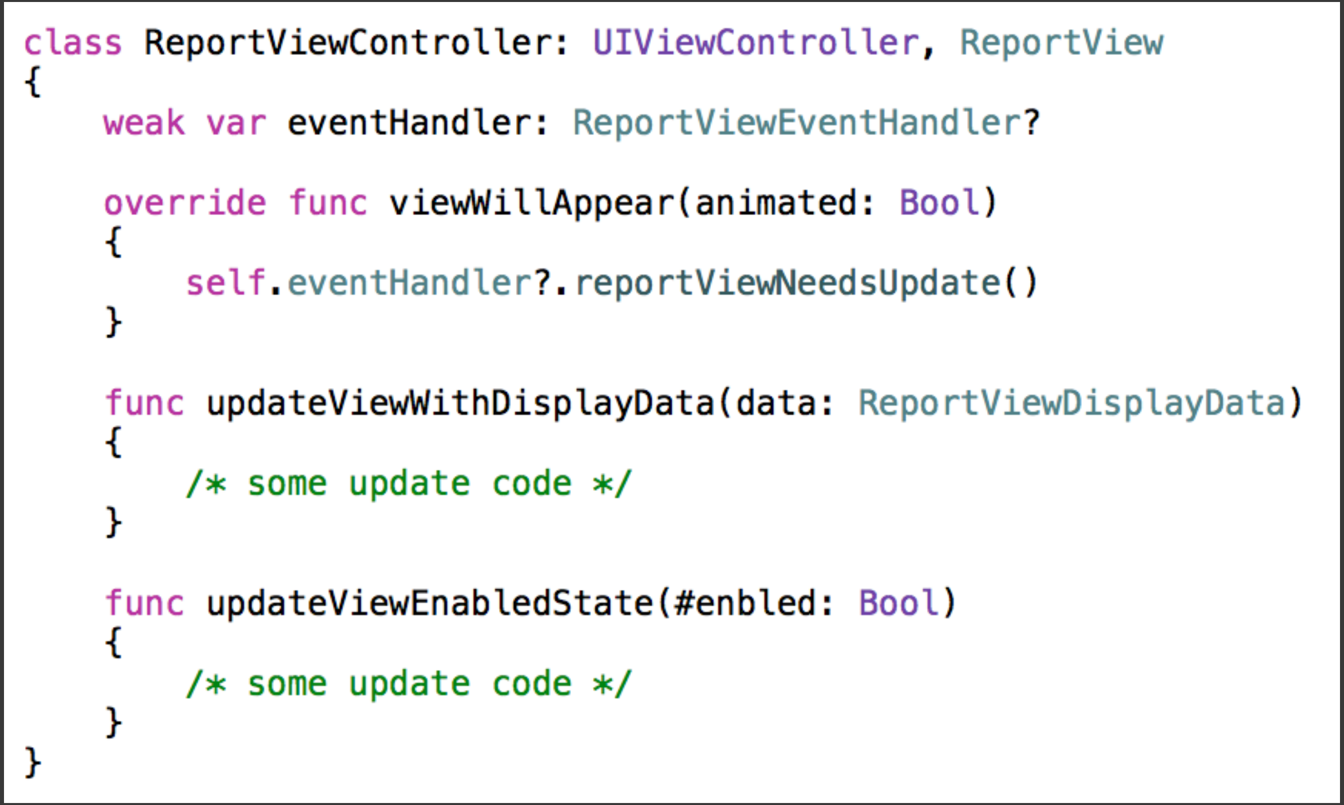
most of the time it consist of an event handler and configuration / update methods

View - imp



```
protocol ReportViewEventHandler: class
{
    func reportViewNeedsUpdate()
    func reportViewDidSelectItemAtIndex(index: Int)
    func reportViewDidSelectDrilldownHeader(index: Int)
}

protocol ReportView
{
    weak var eventHandler: ReportViewEventHandler? { get set }

    func updateViewWithDisplayData(data: ReportViewDisplayData)
    func updateViewEnabledState(#enbled: Bool)
}
```

View - imp



```
class ReportViewController: UIViewController, ReportView
{
    weak var eventHandler: ReportViewEventHandler?

    override func viewWillAppear(animated: Bool)
    {
        self.eventHandler?.reportViewNeedsUpdate()
    }

    func updateViewWithDisplayData(data: ReportViewDisplayData)
    {
        /* some update code */
    }

    func updateViewEnabledState(#enbled: Bool)
    {
        /* some update code */
    }
}
```

View - data types

only basic data types and collections!

String, Int, Float, etc

View - data types

tuples can come really handy!

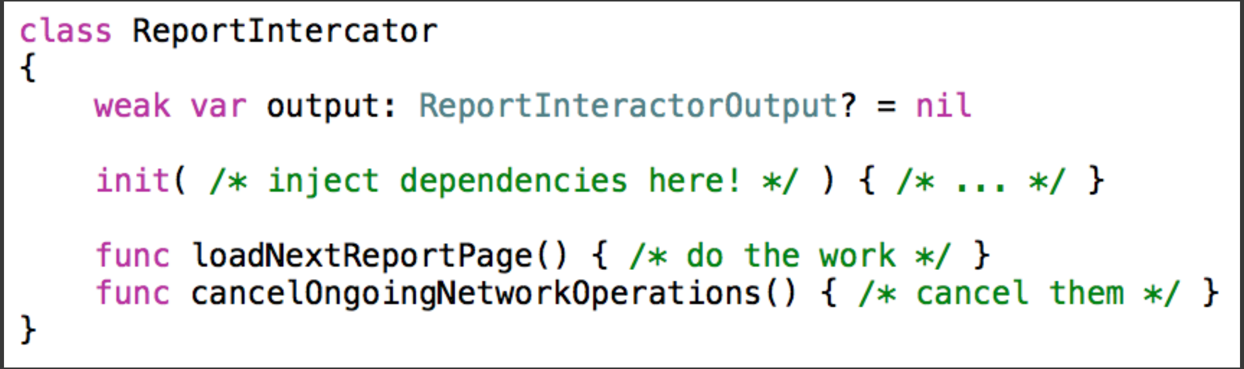
```swift
typealias FilterPresetSelectorItem = (title: String, selected: Bool)
```

Interactor

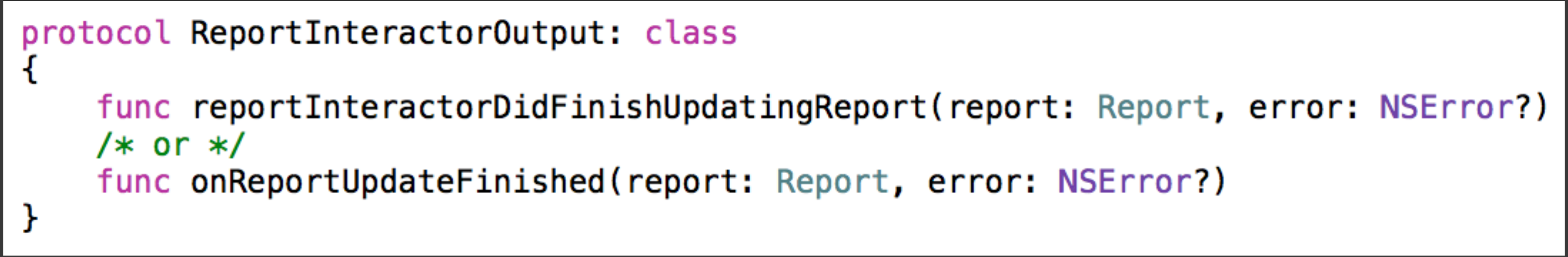
describes a use case / a bunch of use cases

NOTE: my understanding or implementation might differ from the few resources found on the internet!

Interactor - imp



```swift
class ReportIntercator
{
    weak var output: ReportInteractorOutput? = nil

    init( /* inject dependencies here! */ ) { /* ... */ }

    func loadNextReportPage() { /* do the work */ }
    func cancelOngoingNetworkOperations() { /* cancel them */ }
}
```

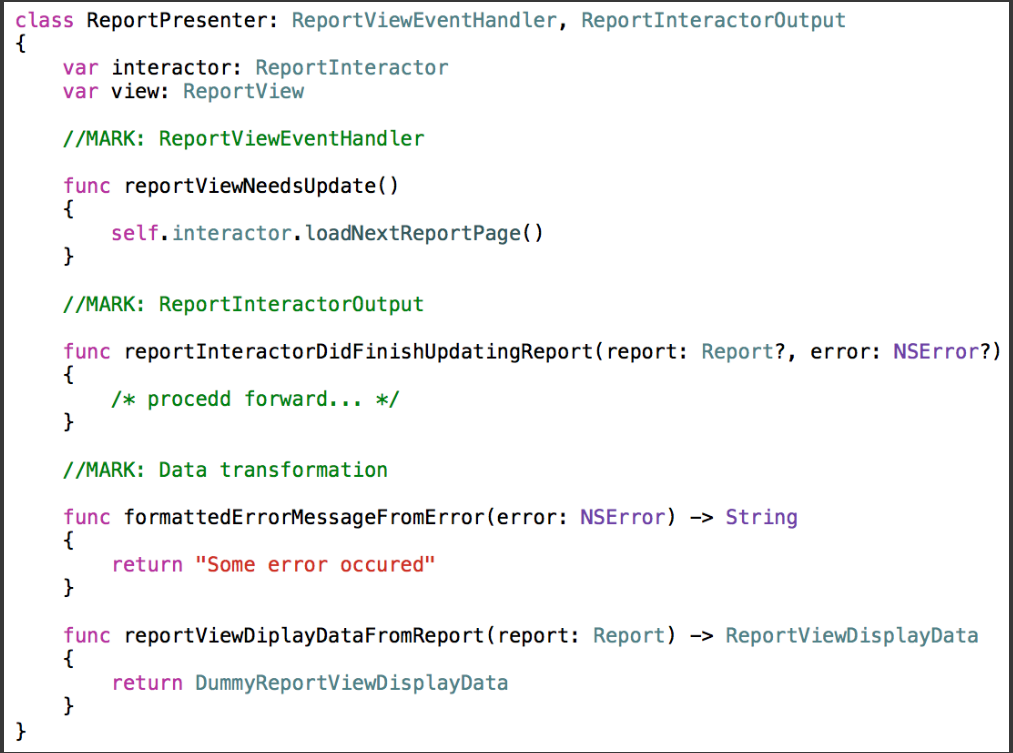


```swift
protocol ReportInteractorOutput: class
{
    func reportInteractorDidFinishUpdatingReport(report: Report, error: NSError?)
    /* or */
    func onReportUpdateFinished(report: Report, error: NSError?)
}
```

Presenter

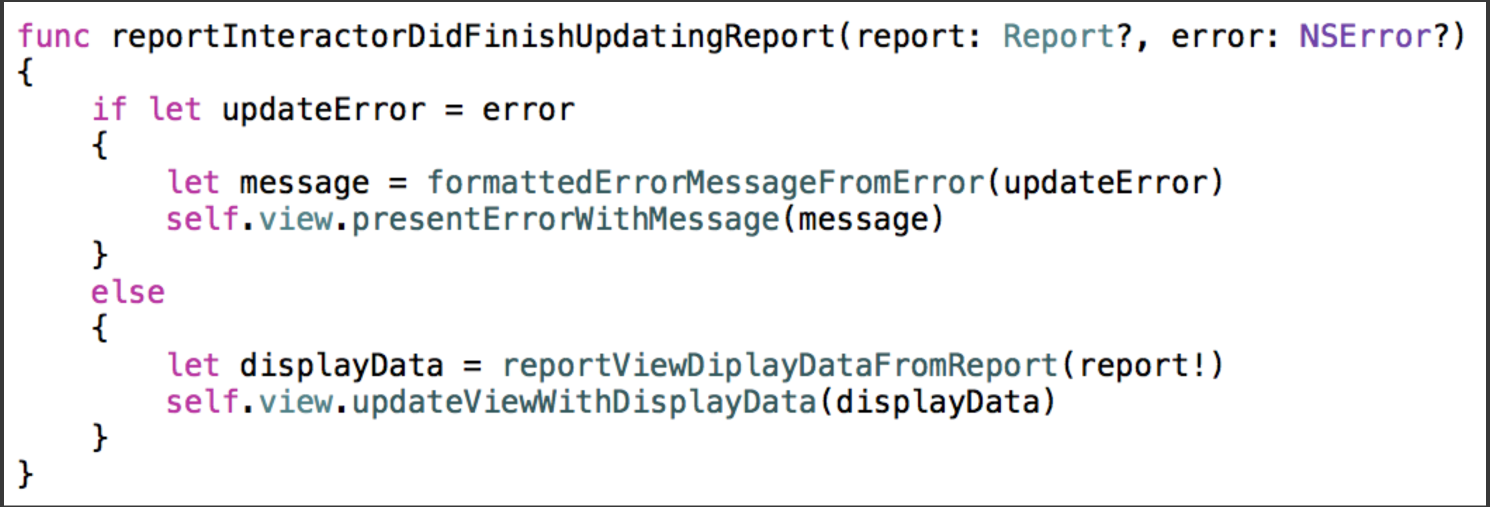
acts as a mediator between view and interactor. translates view events into interactor language / translates interactor output into view data and ask the view to update itself

Presenter - imp



```
class ReportPresenter: ReportViewEventHandler, ReportInteractorOutput
{
    var interactor: ReportInteractor
    var view: ReportView

    //MARK: ReportViewEventHandler

    func reportViewNeedsUpdate()
    {
        self.interactor.loadNextReportPage()
    }

    //MARK: ReportInteractorOutput

    func reportInteractorDidFinishUpdatingReport(report: Report?, error: NSError?)
    {
        /* procedd forward... */
    }

    //MARK: Data transformation

    func formattedErrorMessageFromError(error: NSError) -> String
    {
        return "Some error occured"
    }

    func reportViewDiplayDataFromReport(report: Report) -> ReportViewDisplayData
    {
        return DummyReportViewDisplayData
    }
}
```

Presenter - imp



```
func reportInteractorDidFinishUpdatingReport(report: Report?, error: NSError?)
{
    if let updateError = error
    {
        let message = formattedErrorMessageFromError(updateError)
        self.view.presentErrorWithMessage(message)
    }
    else
    {
        let displayData = reportViewDiplayDataFromReport(report!)
        self.view.updateViewWithDisplayData(displayData)
    }
}
```

Entities

entities are stupid model objects manipulated by the interactor

Entities - imp

swift struct is perfect for describing entities

NOTE: do not use core data only if it is really necessary :)

Router (and Wireframes)

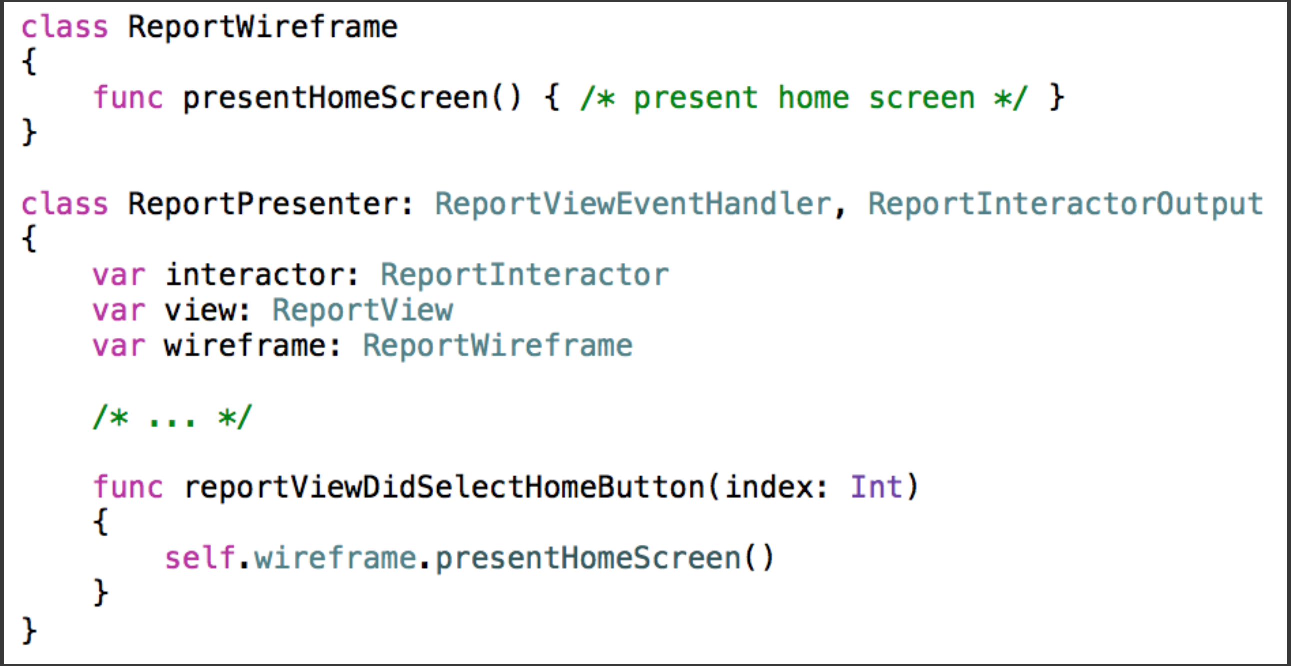
responsible for managing the navigation between screens

(and acts as an entry point / handle for view - presenter - interactor trio, your choice)

Router (and Wireframes)

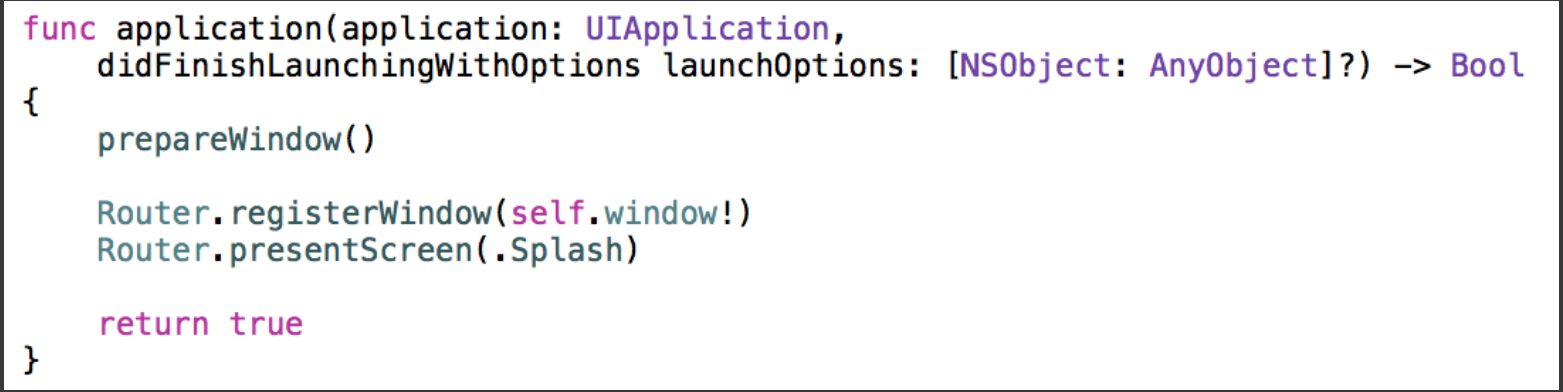
BAD NEWS, the most tricky part...

Router (and Wireframes) - imp



```
class ReportWireframe
{
    func presentHomeScreen() { /* present home screen */ }
}

class ReportPresenter: ReportViewEventHandler, ReportInteractorOutput
{
    var interactor: ReportInteractor
    var view: ReportView
    var wireframe: ReportWireframe

    /* ... */

    func reportViewDidSelectHomeButton(index: Int)
    {
        self.wireframe.presentHomeScreen()
    }
}
```

Router (and Wireframes) - imp

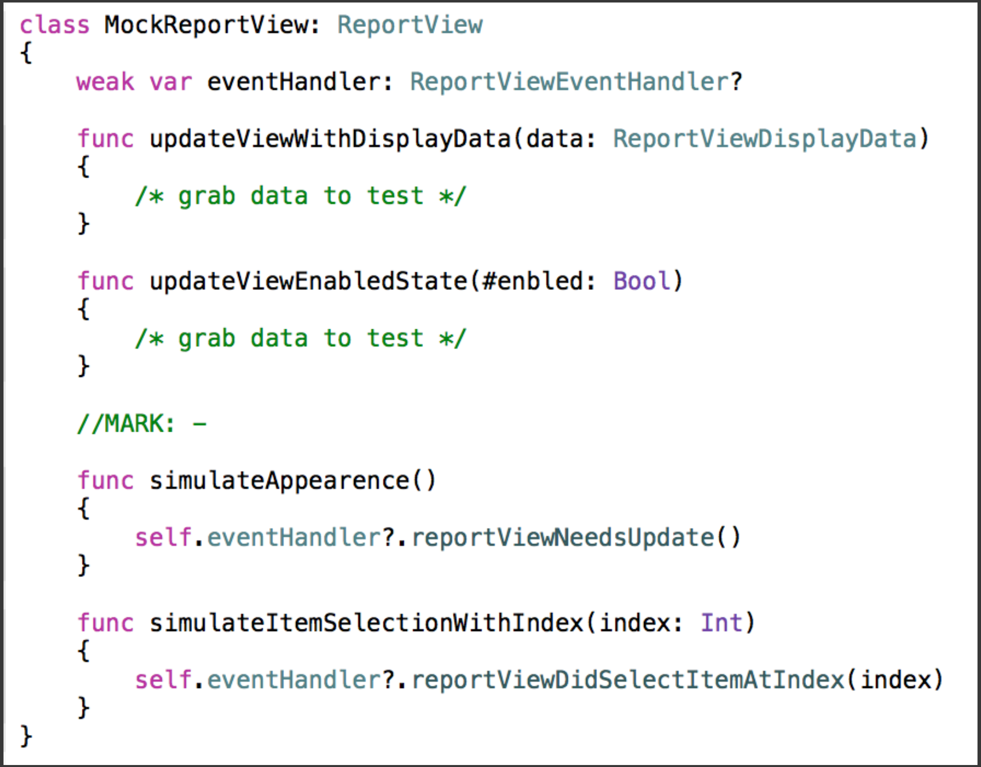


```
func application(application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool
{
    prepareWindow()

    Router.registerWindow(self.window!)
    Router.presentScreen(.Splash)

    return true
}
```

Testing

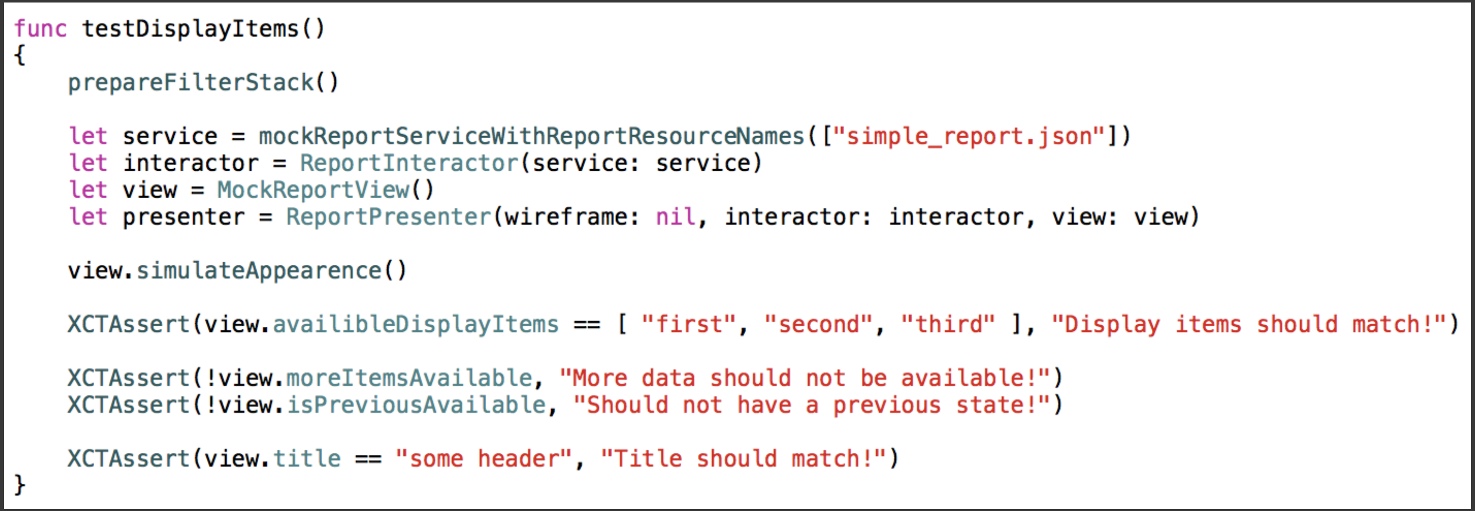
testing should be easy by mocking services / providers

Testing - imp



```
class MockReportView: ReportView
{
    weak var eventHandler: ReportViewEventHandler?

    func updateViewWithDisplayData(data: ReportViewDisplayData)
    {
        /* grab data to test */
    }

    func updateViewEnabledState(#enbled: Bool)
    {
        /* grab data to test */
    }

    //MARK: -

    func simulateAppearence()
    {
        self.eventHandler?.reportViewNeedsUpdate()
    }

    func simulateItemSelectionWithIndex(index: Int)
    {
        self.eventHandler?.reportViewDidSelectItemAtIndex(index)
    }
}
```

Testing - imp



```
func testDisplayItems()
{
    prepareFilterStack()

    let service = mockReportServiceWithReportResourceNames(["simple_report.json"])
    let interactor = ReportInteractor(service: service)
    let view = MockReportView()
    let presenter = ReportPresenter(wireframe: nil, interactor: interactor, view: view)

    view.simulateAppearence()

    XCTAssert(view.availibleDisplayItems == [ "first", "second", "third" ], "Display items should match!")

    XCTAssert(!view.moreItemsAvailable, "More data should not be available!")
    XCTAssert(!view.isPreviousAvailable, "Should not have a previous state!")

    XCTAssert(view.title == "some header", "Title should match!")
}
```

Summary

use VIPER!

remember the most important rules:

KISS, YAGNI

Resources

google ios viper architecture and read everything!

Thank you!

contact me at peter@wearedip.eu