POSSIBLE mobile

Refactoring a Massive View Controller in Swift

Let's Make View Controllers Great Again

by Jeremiah Gage

We’ve all been there…

How the f*** did this view controller get so big?!

Maybe it’s time to refactor?

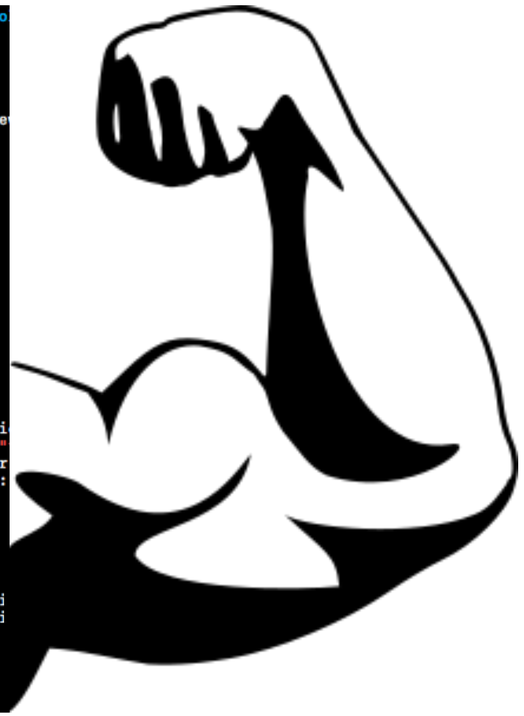
Eh, I only need to add a couple of lines.

I’ll do it later.

Refactoring

KEEPING YOUR CODE HEALTHY

```
class MainViewController: UIViewController, UICollectionViewDataSource, UICo

    private let collectionView: UICollectionView = {
        let flowLayout = UICollectionViewFlowLayout()
        flowLayout.minimumLineSpacing = 0
        flowLayout.minimumInteritemSpacing = 10
        let collectionView = UICollectionView(frame: CGRect(), collectionVie
        return collectionView
    }()

    private var myProfile: [String: AnyObject]?
    private var myTweets: [[String: AnyObject]]?
    private var feedTweets: [[String: AnyObject]]?
    private var followers: [[String: AnyObject]]?

    //MARK: Setup & Teardown

    override func viewDidLoad() {
        super.viewDidLoad()

        view.backgroundColor = UIColor.init(white: 0.9, alpha: 1.0)

        collectionView.translatesAutoresizingMaskIntoConstraints = false
        collectionView.backgroundColor = UIColor.clear()
        collectionView.register(MyProfileCell.self, forCellWithReuseIdentifi
        collectionView.register(TwitCell.self, forCellWithReuseIdentifier: "
        collectionView.register(ProfileCell.self, forCellWithReuseIdentifier
        collectionView.register(TwitHeader.self, forSupplementaryViewOfKind:
        collectionView.dataSource = self
        collectionView.delegate = self
        view.addSubview(collectionView)

        let views = [ "collectionView" : collectionView ]
        var constraints: [NSLayoutConstraint] = []
        constraints.append(contentsOf: NSLayoutConstraint.constraints(withVi
        constraints.append(contentsOf: NSLayoutConstraint.constraints(withVi
        view.addConstraints(constraints)

        title = "Loading..."

        let account = ACAccountStore()
```



1. What?
2. Why?
3. When?
4. How?

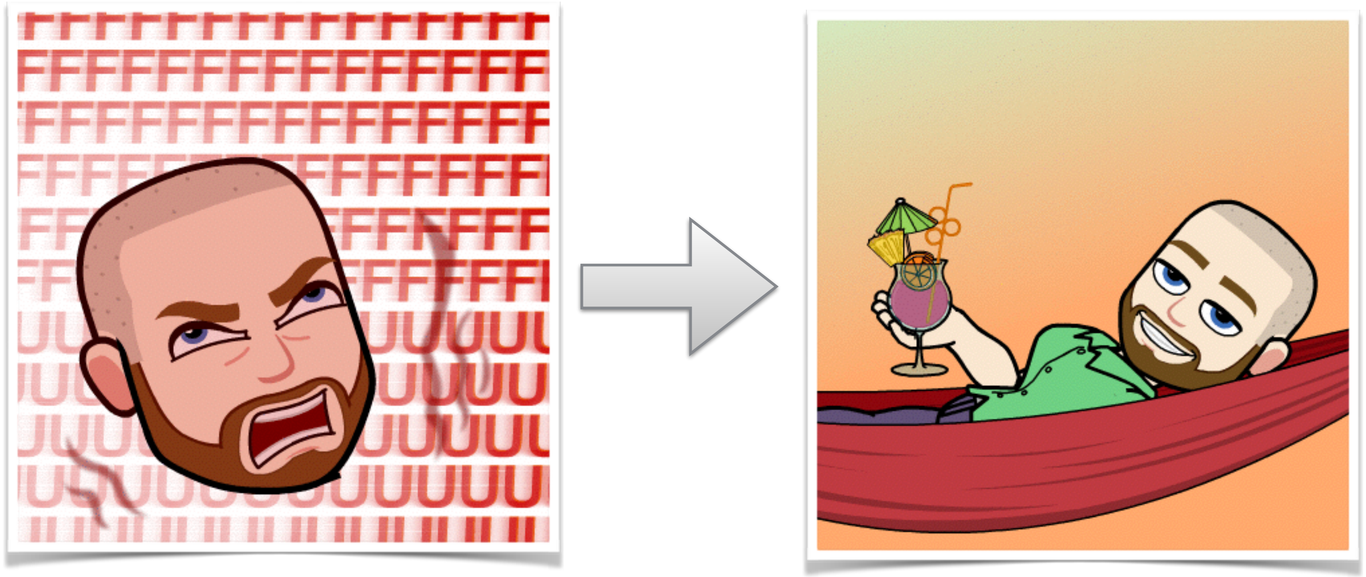
What is refactoring?

Restructuring existing code without changing functionality

Why refactor?

TO KEEP OUR SANITY

- reduce **complexity**
- reduce **risk**
- reduce **stress**
- save **time**



When should you refactor?

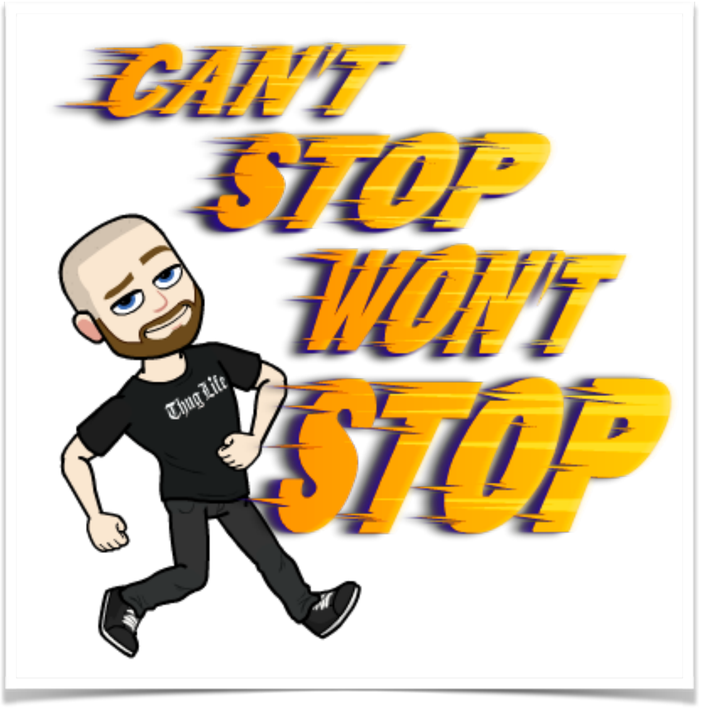
THE FIRST STEP IS ADMITTING YOU HAVE A PROBLEM

- recognize code smells
- if not fun working with the code
- ASAP
  - (if not possible, then document)
- continuously

Continuous Refactoring

ALL DAY ERR DAY

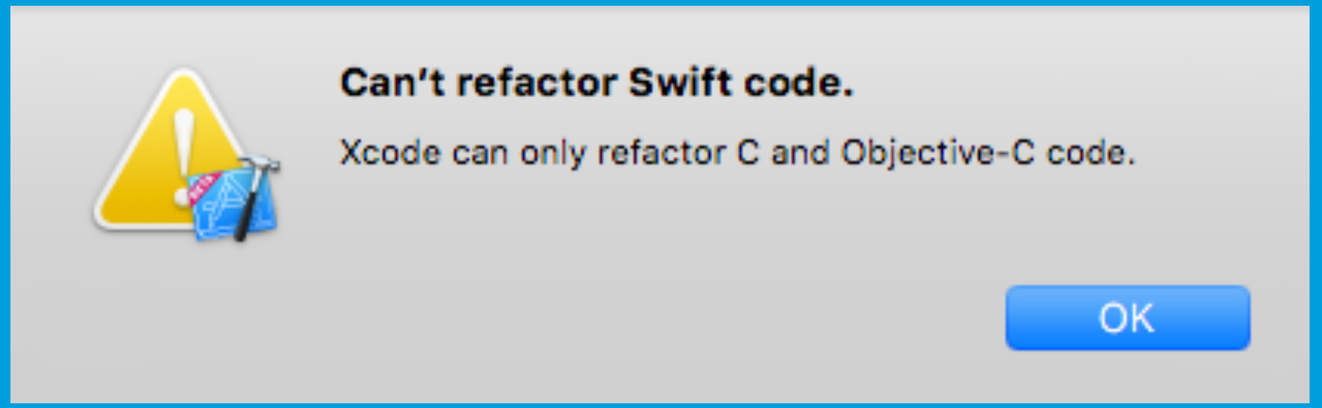
- when touching code
- part of normal development process
- included in estimates
- a skill developed through repetition



Large Refactoring

NOT TALKING ABOUT YO MAMA

- inevitable
- can be risky
- test, test, test!


Can’t refactor Swift code.
Xcode can only refactor C and Objective-C code.
OK

“Programs must be written for people to read, and only incidentally for machines to execute.”

— Abelson and Sussman


Organize your code.

ORGANIZE YOUR CODE | **EXAMPLES**

group related code - before

```
class BeforeGroupRelatedCodeViewController: UIViewController {
    var topView: UIView?
    var bottomView: UIView?

    override func viewDidLoad() {
        super.viewDidLoad()

        topView = UIView()
        if let topView = topView {
            topView.backgroundColor = UIColor.lightGray
            topView.layer.borderWidth = 1
            topView.layer.borderColor = UIColor.black.cgColor
            view.addSubview(topView)
        }

        bottomView = UIView()
        if let bottomView = bottomView {
            bottomView.backgroundColor = UIColor.darkGray
            bottomView.layer.borderWidth = 2
            bottomView.layer.borderColor = UIColor.red.cgColor
            view.addSubview(bottomView)
        }
    }
}
```

ORGANIZE YOUR CODE | **EXAMPLES**

group related code - after

```
class AfterGroupRelatedCodeViewController: UIViewController {
    var topView: UIView = {
        let view = UIView()
        view.backgroundColor = UIColor.lightGray
        view.layer.borderWidth = 1
        view.layer.borderColor = UIColor.black.cgColor
        return view
    }()
    var bottomView: UIView = {
        let view = UIView()
        view.backgroundColor = UIColor.darkGray
        view.layer.borderWidth = 2
        view.layer.borderColor = UIColor.red.cgColor
        return view
    }()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(topView)
        view.addSubview(bottomView)
    }
}
```

ORGANIZE YOUR CODE | **EXAMPLES**

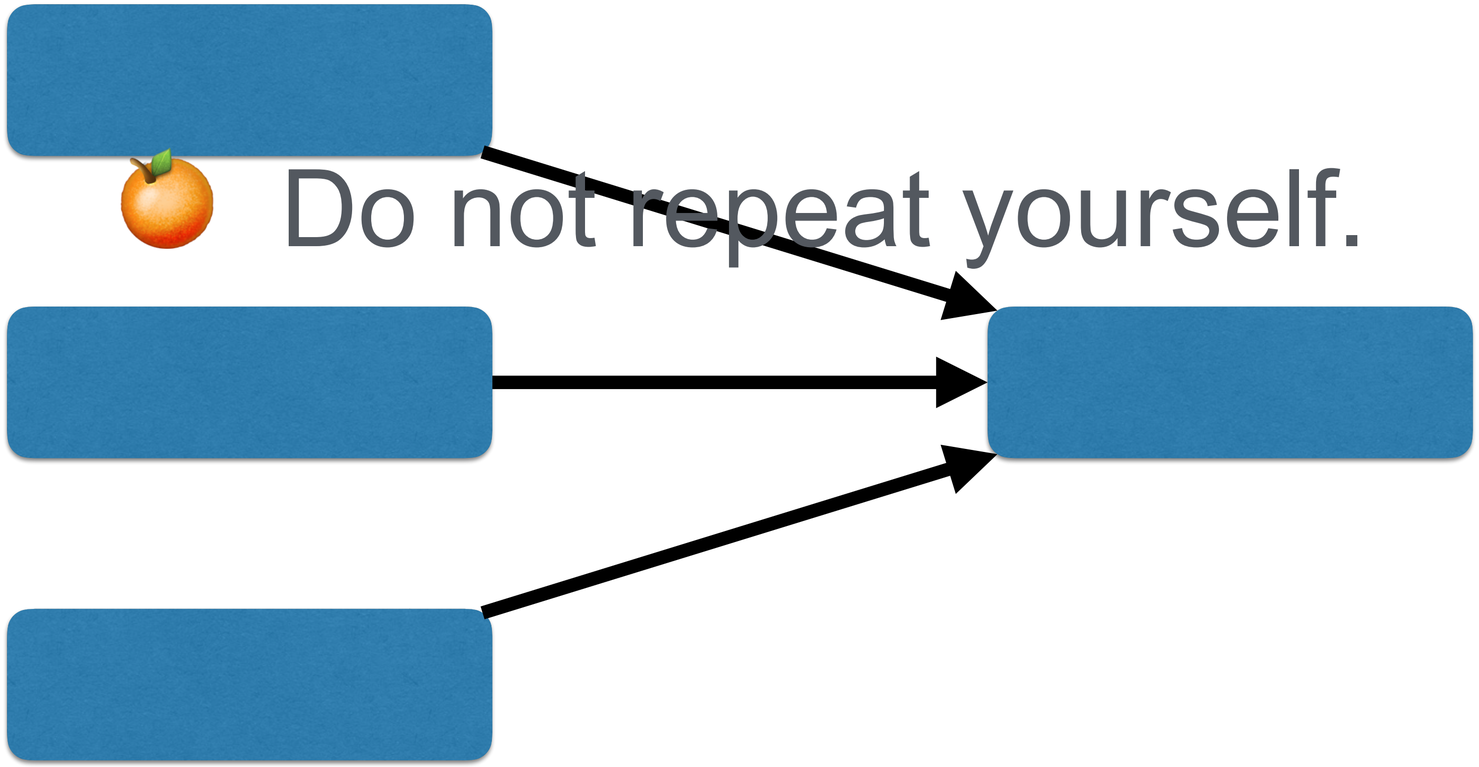
group related methods - before

```swift
class BeforeGroupRelatedMethodsViewController: UIViewController {
    private func setupNotifications() { /* ... */ }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        /* ... */
    }

    @IBAction func button2Tapped() { /* ... */ }

    override func viewDidLoad() {
        super.viewDidLoad()
        /* ... */
    }

    func hideButtons() {
        button1.isHidden = true
        button2.isHidden = true
    }

    private var button1 = UIButton()
    private var button2 = UIButton()

    private func setupTableView() { /* ... */ }

    private func setupButtons() { /* ... */ }

    @IBAction func button1Tapped() { /* ... */ }
}
```

ORGANIZE YOUR CODE | **EXAMPLES**

group related methods - after

```
class AfterGroupRelatedMethodsViewController: UIViewController {
    fileprivate var button1 = UIButton()
    fileprivate var button2 = UIButton()

    //MARK: Setup & Teardown

    override func viewDidLoad() { /* ... */ }
    override func viewDidAppear(_ animated: Bool) { /* ... */ }

    private func setupButtons() { /* ... */ }
    private func setupTableView() { /* ... */ }
    private func setupNotifications() { /* ... */ }

    //MARK: Actions

    @IBAction func button1Tapped() { /* ... */ }
    @IBAction func button2Tapped() { /* ... */ }
}

//MARK: Public

extension AfterGroupRelatedMethodsViewController {
    func hideButtons() {
        button1.isHidden = true
        button2.isHidden = true
    }
}
```


Do not repeat yourself.

★ DO NOT REPEAT YOURSELF | **EXAMPLES**

use a loop - before

```
print("line 1")
print("line 2")
print("line 3")
print("line 4")
print("line 5")
print("line 6")
print("line 7")
print("line 8")
```

★ DO NOT REPEAT YOURSELF | **EXAMPLES**

use a loop - after

```
for i in 1...8 {
    print("line \(i)")
}
```

★ DO NOT REPEAT YOURSELF | **EXAMPLES**

extract to a method - before

```
let redView = UIView()
redView.backgroundColor = UIColor.red
redView.layer.borderColor = UIColor.black.cgColor
redView.layer.borderWidth = 1

let greenView = UIView()
greenView.backgroundColor = UIColor.green
greenView.layer.borderColor = UIColor.black.cgColor
greenView.layer.borderWidth = 2

let blueView = UIView()
blueView.backgroundColor = UIColor.blue
blueView.layer.borderColor = UIColor.black.cgColor
blueView.layer.borderWidth = 3
```

★ DO NOT REPEAT YOURSELF | **EXAMPLES**

extract to a method - after

```
func createView(backgroundColor: UIColor, borderColor: UIColor = UIColor.black, borderWidth: CGFloat = 1) ->
    UIView {
    let view = UIView()
    view.backgroundColor = backgroundColor
    view.layer.borderColor = borderColor.cgColor
    view.layer.borderWidth = borderWidth
    return view
}

let redView2 = createView(backgroundColor: UIColor.red)
let greenView2 = createView(backgroundColor: UIColor.green, borderWidth: 2)
let blueView2 = createView(backgroundColor: UIColor.blue, borderWidth: 3)
```

★ DO NOT REPEAT YOURSELF | **EXAMPLES**

extract to an object (class, struct, enum) - before

```
let yellowLabel = UILabel()
yellowLabel.text = "Yellow"
yellowLabel.backgroundColor = UIColor.yellow
yellowLabel.layer.borderColor = UIColor.black.cgColor
yellowLabel.layer.borderWidth = 1
yellowLabel.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(yellowLabel)
view.addConstraint(NSLayoutConstraint(item: yellowLabel, attribute: .leadingMargin, relatedBy: .
    equal, toItem: view, attribute: .leadingMargin, multiplier: 1, constant: 0))
view.addConstraint(NSLayoutConstraint(item: yellowLabel, attribute: .trailingMargin, relatedBy: .
    equal, toItem: view, attribute: .trailingMargin, multiplier: 1, constant: 0))

let orangeLabel = UILabel()
orangeLabel.text = "Orange"
orangeLabel.backgroundColor = UIColor.orange
orangeLabel.layer.borderColor = UIColor.black.cgColor
orangeLabel.layer.borderWidth = 1
yellowLabel.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(orangeLabel)
view.addConstraint(NSLayoutConstraint(item: orangeLabel, attribute: .leadingMargin, relatedBy: .
    equal, toItem: view, attribute: .leadingMargin, multiplier: 1, constant: 0))
view.addConstraint(NSLayoutConstraint(item: orangeLabel, attribute: .trailingMargin, relatedBy: .
    equal, toItem: view, attribute: .trailingMargin, multiplier: 1, constant: 0))
```

DO NOT REPEAT YOURSELF | **EXAMPLES**

extract to an object (class, struct, enum) - after

class in
separate file

```
let yellowLabel = ColoredLabel(text: "Yellow", color: UIColor.yellow)
yellowLabel.addTo(view: view)
let orangeLabel = ColoredLabel(text: "Orange", color: UIColor.orange)
orangeLabel.addTo(view: view)
```

★ DO NOT REPEAT YOURSELF | **EXAMPLES**

use extensions - before

```
class LightGrayViewController: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        for view in view.subviews {
            view.backgroundColor = UIColor.gray
            view.layer.borderWidth = 1
            view.layer.borderColor = UIColor.black.cgColor
        }
    }
}

class DarkGrayViewController: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        for view in view.subviews {
            view.backgroundColor = UIColor.darkGray
            view.layer.borderWidth = 1
            view.layer.borderColor = UIColor.black.cgColor
        }
    }
}
```

DO NOT REPEAT YOURSELF | **EXAMPLES**

use extensions - after

extension in separate file

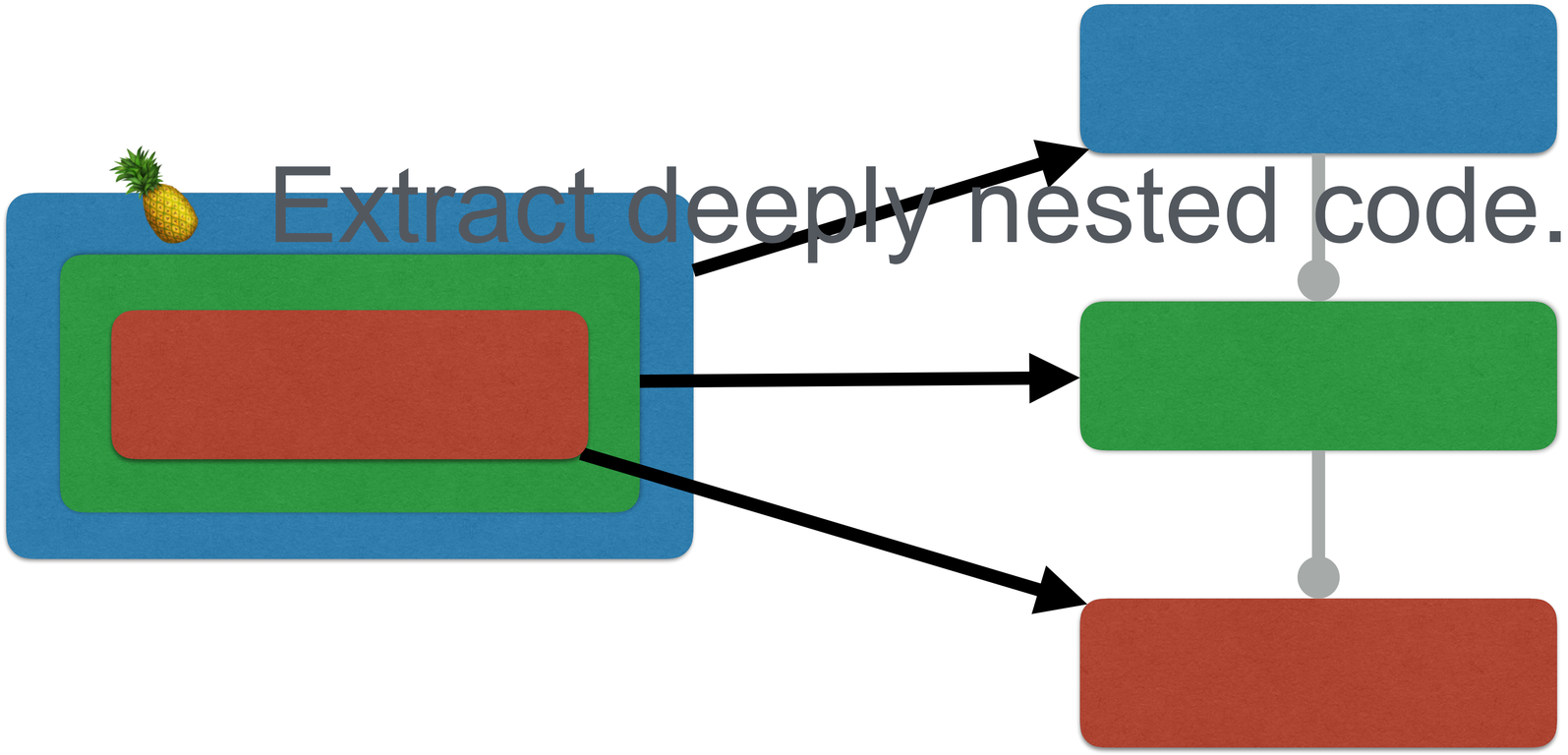
```
class LightGrayViewController2: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        colorViews(color: UIColor.lightGray)
    }
}

class DarkGrayViewController2: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        colorViews(color: UIColor.darkGray)
    }
}
```

Don’t reinvent the wheel.

⭐ DON’T REINVENT THE WHEEL | **EXAMPLES**

- use what Apple gives you
  - e.g. don’t try to make a table view act like a collection view
- use 3rd party libraries
  - can reduce code complexity
  - keep interface limited

Extract deeply nested code.



★ EXTRACT DEEPLY NESTED CODE | **EXAMPLES**

consolidate conditional statements - before

```
var numbers: [Int]? = [1, 2, 3, 4]
if let numbers = numbers {
    if numbers.count > 0 {
        if numbers.count < 5 {
            if numbers[0] == 1 {
                if numbers[1] == 2 {
                    if numbers[2] == 3 {
                        if numbers[3] == 4 {
                            print("numbers are good")
                        }
                    }
                }
            }
        }
    }
}
```

★ EXTRACT DEEPLY NESTED CODE | **EXAMPLES**

consolidate conditional statements - after

```
if let numbers = numbers, numbers.count > 0, numbers.count < 5 {
    let numbersAreGood = numbers[0] == 1 && numbers[1] == 2 && numbers[2] == 3 && numbers[3] == 4
    if numbersAreGood {
        print("numbers are good")
    }
}
```

EXTRACT DEEPLY NESTED CODE | **EXAMPLES**

extract to a method - before

```
var letters: [Character]? = ["A", "B", "C", "D"]
if let letters = letters {
    if letters.count > 0 {
        if letters.count < 5 {
            if letters[0] == "A" {
                if letters[1] == "B" {
                    if letters[2] == "C" {
                        if letters[3] == "D" {
                            print("letters are good")
                        }
                    }
                }
            }
        }
    }
}
```

★ EXTRACT DEEPLY NESTED CODE | **EXAMPLES**

extract to a method - after

```
func lettersAreGood(letters: [Character]) -> Bool {
    return letters[0] == "A" && letters[1] == "B" && letters[2] == "C" && letters[3] == "D"
}

if let letters = letters, letters.count > 0, letters.count < 5 {
    if lettersAreGood(letters: letters) {
        print("letters are good")
    }
}
```

★ EXTRACT DEEPLY NESTED CODE | **EXAMPLES**

use a guard statement - before

```
func colorsMatch(view: UIView?, label: UILabel?, button: UIButton?) -> (Bool, UIColor?) {
    if let view = view {
        if let label = label {
            if let button = button {
                if view.backgroundColor != label.backgroundColor {
                    return (false, nil)
                }
                if view.backgroundColor != button.backgroundColor {
                    return (false, nil)
                }
                if label.backgroundColor != button.backgroundColor {
                    return (false, nil)
                }
                return (true, view.backgroundColor)
            }
        }
    }
    return (false, nil)
}
```

★ EXTRACT DEEPLY NESTED CODE | **EXAMPLES**

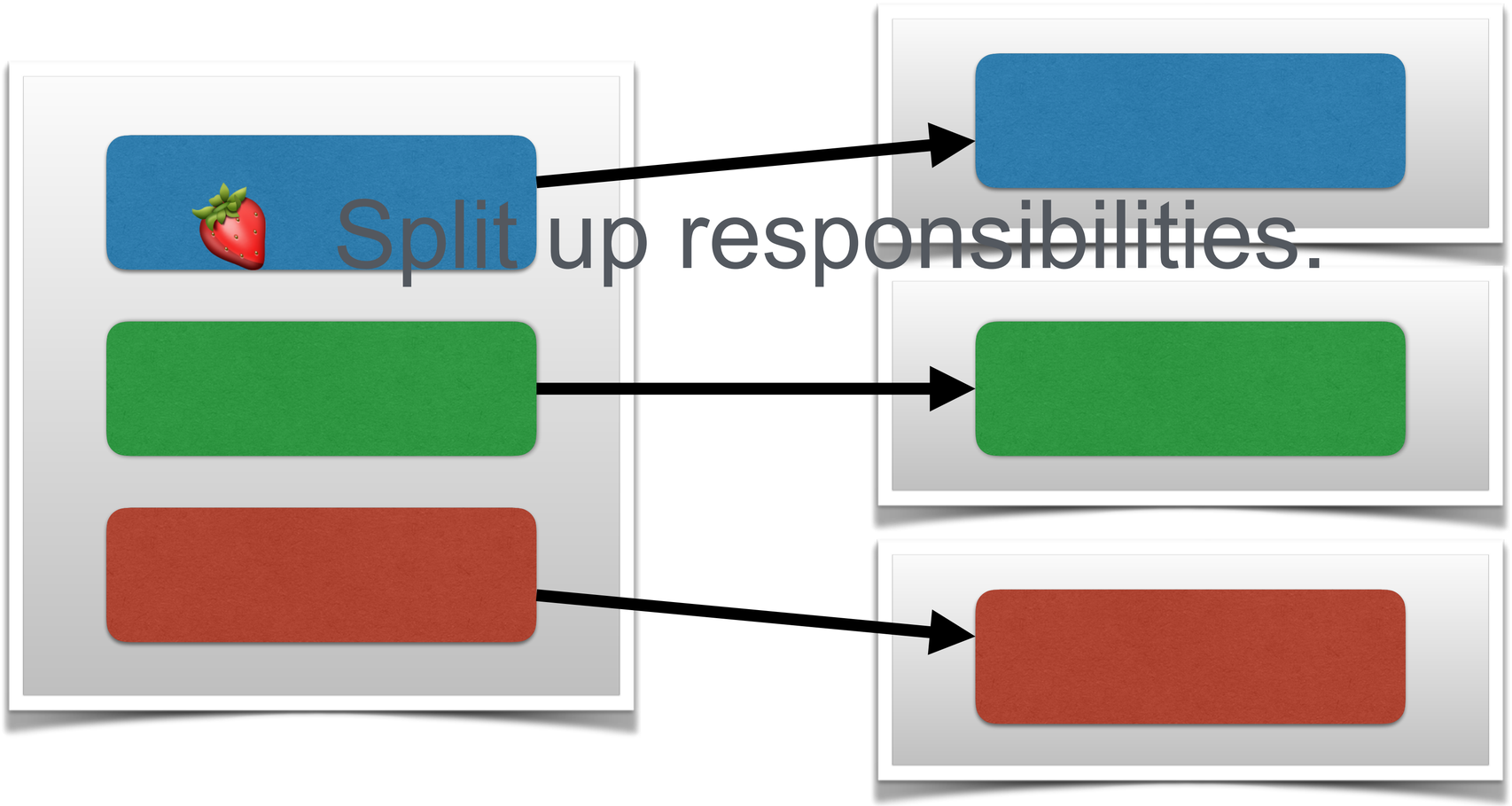
use a guard statement - after

```
func colorsMatch(view: UIView?, label: UILabel?, button: UIButton?) -> (Bool, UIColor?) {
    guard let view = view, let label = label, let button = button,
        view.backgroundColor == label.backgroundColor,
        view.backgroundColor == button.backgroundColor,
        label.backgroundColor == button.backgroundColor
        else {
            return (false, nil)
    }
    return (true, view.backgroundColor)
}
```

 Use one file per class.

⭐ USE ONE FILE PER CLASS | **EXAMPLES**

- use separate files for views and models
- separate structs, enums, protocols, and extensions if necessary
- be aware of access control


Split up responsibilities.

★ SPLIT UP RESPONSIBILITIES | **EXAMPLES**

extract views - before

```
class BeforeExtractViewsViewController: UIViewController {
    var label: UILabel = UILabel()
    var button: UIButton = UIButton()

    override func viewDidLoad() {
        super.viewDidLoad()

        label.translatesAutoresizingMaskIntoConstraints = false
        label.text = "I'm a label"
        label.backgroundColor = UIColor.blue

        button.translatesAutoresizingMaskIntoConstraints = false
        button.setTitle("Click Me", for: .normal)
        button.setTitleColor(UIColor.red, for: .normal)
        button.layer.borderWidth = 1
        button.layer.borderColor = UIColor.black.cgColor

        view.addSubview(label)
        view.addSubview(button)
        let views = ["label" : label, "button" : button] as [String : Any]
        view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: "H:|[label]-[button]-|", options:
            NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
        view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: "V:|-[label]", options:
            NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
        view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: "V:|-[button]", options:
            NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
    }
}
```

SPLIT UP RESPONSIBILITIES | **EXAMPLES**

extract views - after

```
class AfterExtractViewsViewController: UIViewController {
    override func loadView() {
        view = ExtractViewsView(labelText: "I'm a label", buttonTitle: "Click Me")
    }
}
```

view in
separate file

★ SPLIT UP RESPONSIBILITIES | **EXAMPLES**

interfacing with extracted objects

```
protocol ExtractedDelegate: AnyObject {
    func extractedDidSomething(extracted: Extracted)
}

class Extracted {
    weak var delegate: ExtractedDelegate?
    var publicProperty: AnyObject?

    private var privateProperty: AnyObject

    init(something: AnyObject) {
        privateProperty = something
    }

    private func didSomething() {
        delegate?.extractedDidSomething(extracted: self)
    }

    func setValue(for something: AnyObject) {
        privateProperty = something
    }
}
```

★ SPLIT UP RESPONSIBILITIES | **EXAMPLES**

extract models - before

```
class BeforeExtractModelsViewController: UIViewController {
    var meal: [String : String] = [:]
    override func viewDidLoad() {
        super.viewDidLoad()

        meal["mainCourse"] = "salmon"
        meal["side"] = "rice"
        meal["drink"] = "vodka"
        meal["dessert"] = "ice cream"
    }
}
```

★ SPLIT UP RESPONSIBILITIES | **EXAMPLES**

extract models - after

```
class AfterExtractModelsViewController: UIViewController {
    var meal = Meal(mainCourse: "salmon", side: "ride", drink: "vodka", dessert: "ice cream")
}
```

struct in
separate file

SPLIT UP RESPONSIBILITIES | **EXAMPLES**

extract protocol implementations - before

```
class BeforeExtractProtocolImplementationsViewController: UIViewController,
    UICollectionViewDataSource {
    let carCellReuseIdentifier = "carCellReuseIdentifier"
    var carsCollectionView = UICollectionView()

    override func viewDidLoad() {
        super.viewDidLoad()
        carsCollectionView.dataSource = self
        carsCollectionView.register(CarCell.self, forCellWithReuseIdentifier:
            "carCellReuseIdentifier")
    }

    //MARK: UICollectionViewDataSource

    func numberOfSections(in collectionView: UICollectionView) -> Int {
        return 1
    }

    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) ->
        Int {
        return 10
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) ->
        UICollectionViewCell {
        var cell: UICollectionViewCell!
        if let carCell = collectionView.dequeueReusableCell(withReuseIdentifier:
            carCellReuseIdentifier, for: indexPath) as? CarCell {
            carCell.configure(indexPath: indexPath)
            cell = carCell
        }
        return cell
    }
}
```

★ SPLIT UP RESPONSIBILITIES | **EXAMPLES**

extract protocol implementations - after

```
class AfterExtractProtocolImplementationsViewController: UIViewController {
    var carsCollectionView = UICollectionView()
    var carsDataSource: CarsDataSource?

    override func viewDidLoad() {
        super.viewDidLoad()
        carsDataSource = CarsDataSource(collectionView: carsCollectionView)
    }
}
```

class in separate file

★ SPLIT UP RESPONSIBILITIES | **EXAMPLES**

extract services - before

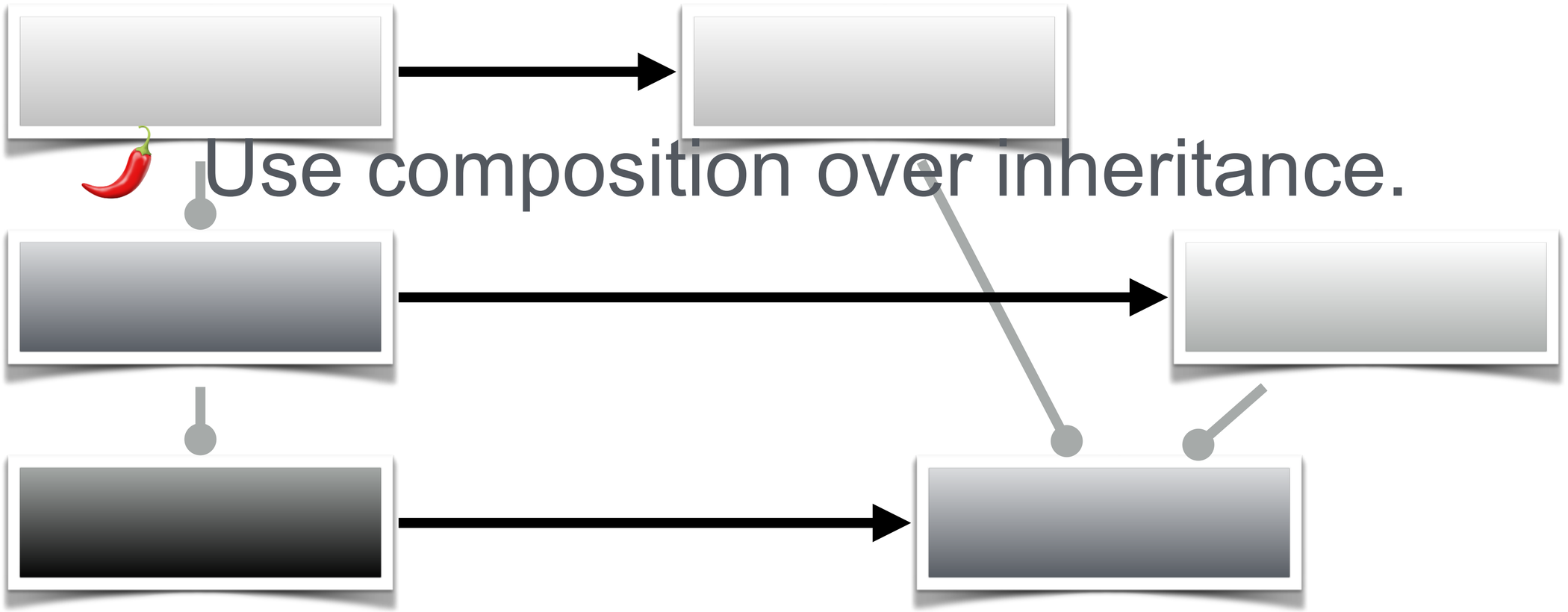
```
class BeforeExtractServicesViewController: UIViewController {
    var userId: Int?
    var userName: String?

    override func viewDidLoad() {
        super.viewDidLoad()

        if let userIdString = UserDefaults.standard.string(forKey: "userId"),
            let userId = Int(userIdString) {
            self.userId = userId
        }
        if let userName = UserDefaults.standard.string(forKey: "userName") {
            self.userName = userName
        }

        print("user id: \(userId)")
        print("user name: \(userName)")
    }
}
```

SPLIT UP RESPONSIBILITIES | **EXAMPLES**

extract services - after

```
class AfterExtractServicesViewController: UIViewController {
    var accountManager = AccountManager()
    override func viewDidLoad() {
        super.viewDidLoad()

        print("user id: \(accountManager.userId)")
        print("user name: \(accountManager.userName)")
    }
}
```

class in separate file


🌶 Use composition over inheritance.

★ USE COMPOSITION OVER INHERITANCE | **EXAMPLES**

protocol oriented programming - before

```
class LoggingViewController: UIViewController {
    func log(output: [String]) {
        for string in output {
            print(string)
        }
    }
}

class DecorateViewsController: LoggingViewController {
    func decorateAllSubviews(color: UIColor) {
        for subview in view.subviews {
            subview.backgroundColor = color
        }
    }
}

class MainViewController: DecorateViewsController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        log(output: view.subviews.map({ $0.description }))
        decorateAllSubviews(color: UIColor.red)
    }
}
```

★ USE COMPOSITION OVER INHERITANCE | **EXAMPLES**

protocol oriented programming - after

```
protocol Loggable {
    func log(output: [String])
}
extension Loggable {
    func log(output: [String]) {
        for string in output {
            print(string)
        }
    }
}

protocol SubviewsDecoratable {
    func decorateAllSubviews(color: UIColor)
}
extension SubviewsDecoratable where Self: UIViewController {
    func decorateAllSubviews(color: UIColor) {
        for subview in view.subviews {
            subview.backgroundColor = color
        }
    }
}

class MainViewController: UIViewController, Loggable, SubviewsDecoratable {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        log(output: view.subviews.map({ $0.description }))
        decorateAllSubviews(color: UIColor.red)
    }
}
```

🍆 Remove unused code.

★ REMOVE UNUSED CODE | **EXAMPLES**

comments and line breaks - before

```
//print 1 through 10
for i in 1...10 {

    print(i)
}

//determine if a label should be created
let createLabel = true

//create a label and add it to the view if necessary
if createLabel {

    //create the label
    let label = UILabel()

    //add the label to the view
    view.addSubview(label)
}
```

★ REMOVE UNUSED CODE | **EXAMPLES**

comments and line breaks - after

```
for i in 1...10 {
    print(i)
}

let createLabel = true
if createLabel {
    let label = UILabel()
    view.addSubview(label)
}
```

★ REMOVE UNUSED CODE | **EXAMPLES**

code that is never executed - before

```
class BeforeCodeNeverExecutedViewController: UIViewController {
    var button1: UIButton!
    var button2: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

//          button1 = UIButton()
//          button1.setTitle("Do Not Touch", for: .normal)
//          button1.addTarget(self, action: #selector(button1Touched), for: .touchUpInside)
//          view.addSubview(button1)

        button2 = UIButton()
        button2.setTitle("Do Not Touch", for: .normal)
        button2.addTarget(self, action: #selector(button2Touched), for: .touchUpInside)
        view.addSubview(button2)
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
//          button1.hidden = true
    }

    func button1Touched() {
        print("button 1 touched")
    }

    func button2Touched() {
        print("button 2 touched")
    }
}
```

★ REMOVE UNUSED CODE | **EXAMPLES**

code that is never executed - after

```
class AfterCodeNeverExecutedViewController: UIViewController {
    var button2: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        button2 = UIButton()
        button2.setTitle("Do Not Touch", for: .normal)
        button2.addTarget(self, action: #selector(button2Touched), for: .touchUpInside)
        view.addSubview(button2)
    }

    func button2Touched() {
        print("button 2 touched")
    }
}
```

★ REMOVE UNUSED CODE | **EXAMPLES**

generated code - before

```
class BeforeGeneratedCodeViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
        title = "With Generated Code"
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    /*
     // MARK: - Navigation

     // In a storyboard-based application, you will often want to do a little preparation before navigation
     override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
     // Get the new view controller using segue.destinationViewController.
     // Pass the selected object to the new view controller.
     }
     */

}
```

★ REMOVE UNUSED CODE | **EXAMPLES**

generated code - after

```
class AfterGeneratedCodeViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        title = "Without Generated Code"
    }
}
```

★ REMOVE UNUSED CODE | **EXAMPLES**

other: imports, reference to self, type declarations, etc. - before

```swift
import UIKit
import AVFoundation
import SafariServices

enum Country: String {
    case USA
    case Canada
    case Brazil
}

class BeforeOtherViewController: UIViewController {
    var selectedCountry: Country = Country.USA
    var validCountries: [Country] = [Country.USA, Country.Canada]
    var countryString: String = ""

    override func viewDidLoad() {
        super.viewDidLoad()

        self.validCountries.removeLast()
        self.validCountries.append(Country.Brazil)
        self.countryString = { () -> [String] in
            var countriesArray: [String] = []
            for country: Country in self.validCountries {
                countriesArray.append(country.rawValue)
            }
            return countriesArray
        }().joined(separator: ",")
    }
}
```

REMOVE UNUSED CODE | **EXAMPLES**

other: imports, reference to self, type declarations, etc. - after

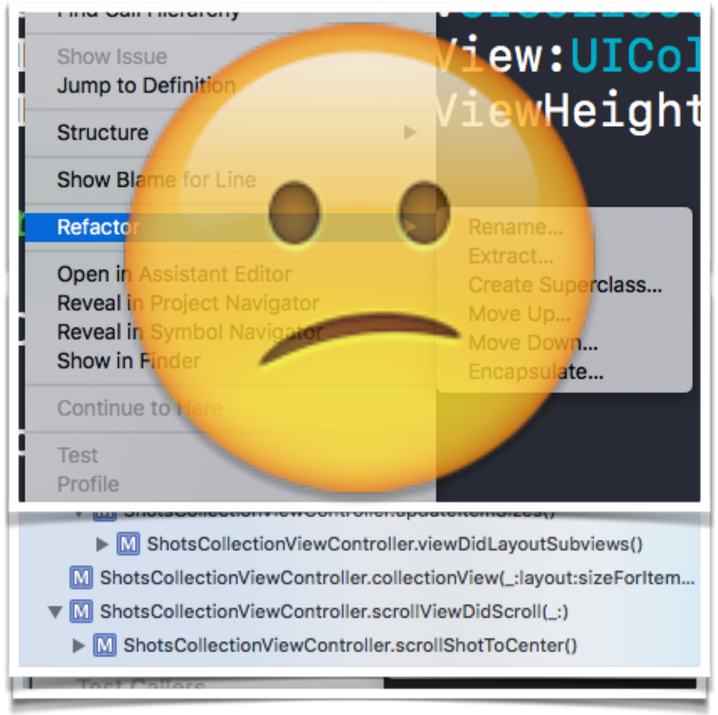
```swift
import UIKit

class AfterOtherViewController: UIViewController {

    var selectedCountry = Country.USA
    var validCountries: [Country] = [.USA, .Canada]
    var countryString = ""

    override func viewDidLoad() {
        super.viewDidLoad()

        validCountries.removeLast()
        validCountries.append(Country.Brazil)
        countryString = validCountries.map({ $0.rawValue }).joined(separator: ",")
    }
}
```

enum in separate file

Navigating Code

IN XCODE

- Open quickly (⌘-shift + O)
- Document items (control + 6)
- Related items (control + 1)
- Find call hierarchy (in context menu)
- Automated refactoring



“Make everything as simple as possible, but not simpler.”

— Albert Einstein

Pitfalls of Refactoring

SUBTITLE INTENTIONALLY LEFT BLANK

- be clear not clever
- don’t abstract too far
- account for performance
- manage time vs. benefit

Recap

VIEW CONTROLLERS, YEAH!

- refactor to reduce code complexity
- continuously refactor
- keep improving
- do what works for you and your project

POSSIBLE mobile

Slides and code available at **https://github.com/jeremiahgage/ViewControllerRefactoring**

Refactoring a Massive View Controller in Swift

Let's Make View Controllers Great Again

@JeremiahGage
Jeremiah.Gage@possible.com
http://PossibleMobile.com