

# Optimizing Swift code for separation of concerns and simplicity

関心の分離と単純化のためのSwiftコードの最適化

# Intro

- Pebble
- Twitter (Fabric)



# Coding Principles

## コーディングの原則

# Coding Principles

## コーディングの原則

- **Simplicity** / 単純
- **Conciseness** / 簡潔
- **Clarity** / 明確

# Separation of Concerns

## 関心の分離

- Code is read much more often than it is written  
コードは書かれることよりも読まれることの方が多い
- Separating the "*what*" from the "*how*"  
*How* から *what* を切り離す

## Example 1 (Before)

```
func textView(_ textView: UITextView, shouldChangeTextIn range: NSRange,
              replacementText text: String) -> Bool {
    sendButton.isEnabled =
        textView.text.utf16.count
        + text.utf16.count
        - range.length <= 140

    return true
}
```

# Example 1 (After)

```
private extension String {
    var characterCountUsingBackendPolicy: Int {
        return utf16.count
    }
}

func textView(_ textView: UITextView,
              shouldChangeTextIn range: NSRange,
              replacementText text: String) -> Bool {
    let characters =
        textView.text.characterCountUsingBackendPolicy
        + text.characterCountUsingBackendPolicy
        - range.length

    let characterLimit = 140

    sendButton.isEnabled = characters <= characterLimit

    return true
}
```

## Example 2 (Before)

```
api.requestReplies(postID: 4815162342) { [weak self] result in
    switch result {
    case .success(let replies):
        var filteredReplies: [Reply] = []
        for reply in replies {
            if !user.isBlocking(reply.author) {
                filteredReplies.append(reply)
            }
        }

        self?.replies = filteredReplies
    case .failure:
        // ...
    }
}
```



## Example 2 (After)

```
extension Collection where Element == Reply {  
    var filteringBlockedContent: [Reply] {  
        return filter { !user.isBlocking($0.author) }  
    }  
}
```

```
api.requestReplies(postID: 4815162342) { [weak self] result in  
    switch result {  
    case .success(let replies):  
        self?.replies = replies.filteringBlockedContent  
    case .failure:  
        // ...  
    }  
}
```

## Example 3 (Before)

```
NSLayoutConstraint.activate([
    subview.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: insets.left),
    subview.topAnchor.constraint(equalTo: view.topAnchor, constant: insets.top),
    view.trailingAnchor.constraint(equalTo: subview.trailingAnchor, constant: insets.right),
    view.bottomAnchor.constraint(equalTo: subview.bottomAnchor, constant: insets.bottom)
])
```

## Example 3 (After)

```
NSLayoutConstraint.activate([
    subview.leadingAnchor ≈ view.leadingAnchor + insets.left,
    subview.topAnchor ≈ view.topAnchor + insets.top,
    view.trailingAnchor ≈ subview.trailingAnchor + insets.right,
    view.bottomAnchor ≈ subview.bottomAnchor + insets.bottom
])
```

```
infix operator ≈ : LayoutAnchorPrecedence
```

```
func ≈ <AnchorType>(lhs: NSLayoutAnchor<AnchorType>,
                    rhs: LayoutAnchorTransform<AnchorType>) -> NSLayoutConstraint {
    return lhs.constraint(equalTo: rhs.anchor, constant: rhs.constant)
}
```

```
func + <AnchorType>(lhs: NSLayoutAnchor<AnchorType>,
                   rhs: CGFloat) -> LayoutAnchorTransform<AnchorType> {
    return LayoutAnchorTransform(anchor: lhs, constant: rhs)
}
```

## Example 3 (After 2)

```
NSLayoutConstraint.activate(NSLayoutConstraint.anchoring(subview, within: view))

extension NSLayoutConstraint {
    static func anchoring(_ subview: UIView,
                          within view: UIView,
                          insets: UIEdgeInsets = .zero) -> [NSLayoutConstraint] {
        return [
            subview.leadingAnchor ≈ view.leadingAnchor + insets.left,
            subview.topAnchor ≈ view.topAnchor + insets.top,
            view.trailingAnchor ≈ subview.trailingAnchor + insets.right,
            view.bottomAnchor ≈ subview.bottomAnchor + insets.bottom
        ]
    }
}
```

## Example 3 (Before and After)

// Before

```
NSLayoutConstraint.activate([
    subview.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: insets.left),
    subview.topAnchor.constraint(equalTo: view.topAnchor, constant: insets.top),
    view.trailingAnchor.constraint(equalTo: subview.trailingAnchor, constant: insets.right),
    view.bottomAnchor.constraint(equalTo: subview.bottomAnchor, constant: insets.bottom)
])
```

// After

```
NSLayoutConstraint.activate(NSLayoutConstraint.anchoring(subview, within: view))
```

# Example 4

```
class MyView: UIView {
    var activeConstraints: [NSLayoutConstraint] = [] {
        didSet { NSLayoutConstraint.deactivate(activeConstraints) }
        didSet { NSLayoutConstraint.activate(activeConstraints) }
    }

    var headerVisible: Bool { didSet {
        activeConstraints = [
            view.topAnchor ≈ (headerVisible ? header.topAnchor : header.bottomAnchor),
            // more ...
        ]
    }}
}
```

## Example 5 (Before)

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return 3
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "identifier", for: indexPath)

    if indexPath.row == 0 {
        cell.textLabel.text = "General"
    } else if indexPath.row == 1 {
        cell.textLabel.text = "Notifications"
    } else if indexPath.row == 2 {
        cell.textLabel.text = "Log Out"
    }

    return cell
}
```

# Example 5 (After)

```
enum Row {
    case general, notifications, logout

    var title: String {
        switch self {
            case .general: return "General"
            // ...
        }
    }
}

let rows: [Row] = [.general, .notifications, .logout]

func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return rows.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = collectionView.dequeueReusableCell(...
    cell.textLabel.text = rows[indexPath.row].title
    return cell
}
```



## Example 6 (Before)

```
func showTutorial() {  
    guard !UserDefaults.standard.bool(forKey: "has_seen_tutorial") else { return }  
  
    // Show tutorial  
  
    UserDefaults.standard.set(true, forKey: "has_seen_tutorial")  
}
```

## Example 6 (After)

```
var hasSeenTutorial: Bool {  
    get { return UserDefaults.standard.bool(forKey: "has_seen_tutorial") }  
    set { UserDefaults.standard.set(newValue, forKey: "has_seen_tutorial") }  
}  
  
func showTutorial() {  
    guard !hasSeenTutorial else { return }  
  
    // Show tutorial  
  
    hasSeenTutorial = true  
}
```

## Example 7 (Before)

```
if #available(iOS 11.0, *) {  
    constraints = [  
        subview.topAnchor ≈ view.safeAreaLayoutGuide.topAnchor,  
        subview.leadingAnchor ≈ view.safeAreaLayoutGuide.leadingAnchor,  
        subview.bottomAnchor ≈ view.safeAreaLayoutGuide.bottomAnchor,  
        subview.trailingAnchor ≈ view.safeAreaLayoutGuide.trailingAnchor,  
    ]  
} else {  
    constraints = [  
        subview.topAnchor ≈ view.topAnchor,  
        subview.leadingAnchor ≈ view.leadingAnchor,  
        subview.bottomAnchor ≈ view.bottomAnchor,  
        subview.trailingAnchor ≈ view.trailingAnchor,  
    ]  
}
```

# Example 7 (After)

```
extension UIView {
    var tw_safeAreaLayoutGuide: UILayoutGuide {
        if #available(iOS 11.0, *) {
            return safeAreaLayoutGuide
        }

        if let tw_safeAreaLayoutGuide = objc_getAssociatedObject(self, associatedKey) as? UILayoutGuide {
            return tw_safeAreaLayoutGuide
        }

        let tw_safeAreaLayoutGuide = UILayoutGuide()
        addLayoutGuide(tw_safeAreaLayoutGuide)

        NSLayoutConstraint.activate(NSLayoutConstraint.anchoring(tw_safeAreaLayoutGuide, within: self))

        objc_setAssociatedObject(self, associatedKey, tw_safeAreaLayoutGuide, .OBJC_ASSOCIATION_RETAIN_NONATOMIC)

        return tw_safeAreaLayoutGuide
    }
}
```

## Example 8 (Before)

```
// Set size big enough to make it easy to tap  
button.frame = CGRect(origin: .zero, size: CGSize(width: 44, height: 44))
```

# Example 8 (After)

```
class TWMinimumHitAreaButton: UIButton {
    override func hitTest(_ point: CGPoint, with event: UIEvent?) -> UIView? {
        guard !isHidden && enabled &&.isUserInteractionEnabled && alpha >= 0.01 else {
            return nil
        }

        let lengthOfTappableAreaOutside: CGFloat = 10.0
        let minimumHitAreaSize = CGSize(width: 44, height: 44)

        // Increase the hit frame to be at least as big as `minimumHitArea`
        let buttonSize = bounds.size
        let widthToAdd = max(minimumHitAreaSize.width - buttonSize.width, lengthOfTappableAreaOutside * 2)
        let heightToAdd = max(minimumHitAreaSize.height - buttonSize.height, lengthOfTappableAreaOutside * 2)
        let largerFrame = bounds.insetBy(dx: -widthToAdd / 2, dy: -heightToAdd / 2)

        // Perform hit test on larger frame
        let hit = largerFrame.contains(point)
        return hit ? self : nil
    }
}
```

# Example 9 (Before)

```
/// [ Send ] / [ Close ] / [ Send ] - [ Close ]
```

```
private let sendButton: UIButton
var showSendButton: Bool {
    didSet {
        if showSendButton {
            addSubview(sendButton)
        } else {
            sendButton.removeFromSuperview()
        }
        configureConstraints()
    }
}
```

```
private let closeButton: UIButton
var showCloseButton: Bool {
    didSet {
        // ...
    }
}
```

## Example 9 (After)

```
private let sendButton: UIButton
private let closeButton: UIButton
private let stackView = UIStackView(arrangedViews: sendButton, closeButton)

var showSendButton: Bool {
    didSet {
        sendButton.isHidden = !showSendButton
    }
}

var showCloseButton: Bool {
    didSet {
        closeButton.isHidden = !showCloseButton
    }
}
```



# Example 10

## Example 10

- ProfileViewController
  - user details
  - children views

## Example 10

- ProfileViewController
  - userID
  - *(user details)*
    - *(children views)*

## Example 10

```
final class ProfileViewController: UIViewController {  
    // Before:  
    private let userInfo: UserInfo  
    private let headerView: ProfileHeaderView  
  
    // After:  
    private var userInfo: UserInfo?  
    private var headerView: ProfileHeaderView?  
    private var spinner: UIActivityIndicatorView?  
    private var retryButton: UIButton?  
}
```

## Example 10

```
final class ProfileViewController: UIViewController {  
    private enum State {  
        case pending  
        case loading(spinner: UIActivityIndicatorView)  
        case failed(retryButton: UIButton)  
        case loaded(userInfo: UserInfo, headerView: ProfileHeaderView)  
    }  
  
    private var state: State = .pending  
}
```

# Example 10

```
final class ProfileViewController: UIViewController {
    var state: State = .pending

    private func loadUserDetails() {
        state = .loading(spinner: UIActivityIndicatorView())

        api.requestUserDetails(userID) { [weak self] result in
            switch result {
            case let .success(userInfo):
                self?.state = .loaded(userInfo: userInfo, headerView: createHeaderView(userInfo))
            case let .failure:
                self?.state = .failed(createRetryButton())
            }
        }
    }
}
```

# Example 10

```
final class ProfileViewController: UIViewController {
    var visibleView: UIView? {
        didSet { visibleView?.removeFromSuperview() }
        didSet { if let visibleView = visibleView { view.addSubview(visibleView) } }
    }
    var state: State = .pending {
        didSet {
            switch state {
            case .pending: visibleView = nil
            case .loading(let spinner): visibleView = spinner
            case .loaded(_, let profileHeaderView): visibleView = profileHeaderView
            case .failed(let retryButton): visibleView = retryButton
            }
        }
    }
}
```

# Summary

- Value in optimizing readability in local scopes  
ローカルスコープの可読性の最適化には価値がある
- DRY (Don't Repeat Yourself)
- Swift enums are awesome



# Happy Swiftting!

Thank you.