SOLID Swift

Joel Márquez

Sr. iOS Software Engineer @ Mercado Libre

@joelmarquez90

@joelmarquez

SwiftBA

- S
- O
- L
- I
- D

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

“Uncle” Bob Martin https://en.wikipedia.org/wiki/Robert_C._Martin

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Una clase debe tener una sola razón para cambiar.

```swift
public class SectionDataSourceRemoveProcessor {
    // MARK: Private properties

    private let currentDataSource: [Section]

    // MARK: Init

    public init(_ currentDataSource: [Section]) {
        self.currentDataSource = currentDataSource
    }

    // MARK: Public methods

    public func remove(_ section: Section) -> [Section] {
        var updatedDataSource = Array(currentDataSource)

        updatedDataSource.removeAll(where: { $0.id == section.id })

        return updatedDataSource
    }
}
```

- Single Responsibility Principle
- **O**pen-Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Las clases deben estar abiertas para la extensión, pero cerradas para la modificación.

Los protocolos deben estar abiertos para la extensión, pero cerrados para la modificación.

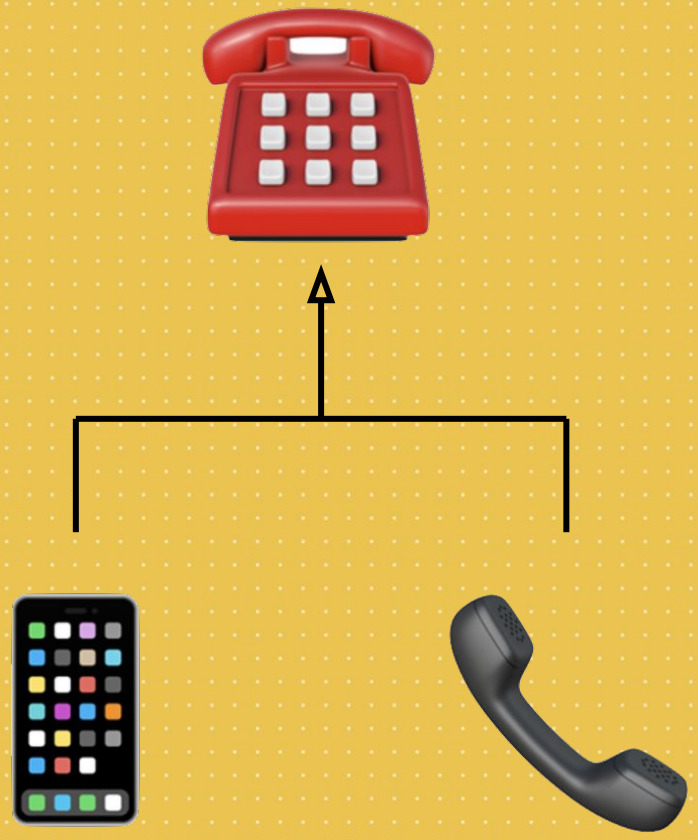
```swift
public protocol CodingStrategy {
    /**
     Encodes a value of type `T: Encodable` into a `Data`.
     */
    func encode<T>(_ value: T) throws -> Data where T: Encodable

    /**
     Decodes a data into a value of type `T: Decodable`.
     */
    func decode<T>(_ type: T.Type, from data: Data) throws -> T where T: Decodable
}
```

```swift
public class JSONCodingStrategy: CodingStrategy {
    public init() {}

    /**
     Encodes a value of type `T: Encodable` into a `Data`.
     */
    public func encode<T>(_ value: T) throws -> Data where T: Encodable {
        return try JSONEncoder().encode(value)
    }

    /**
     Decodes a data into a value of type `T: Decodable`.
     */
    public func decode<T>(_ type: T.Type, from data: Data) throws -> T where T: Decodable {
        return try JSONDecoder().decode(type, from: data)
    }
}
```

```swift
public class PropertyListCodingStrategy: CodingStrategy {
    public init() {}

    /**
     Encodes a value of type `T: Encodable` into a `Data`.
     */
    public func encode<T>(_ value: T) throws -> Data where T: Encodable {
        return try PropertyListEncoder().encode(value)
    }

    /**
     Decodes a data into a value of type `T: Decodable`.
     */
    public func decode<T>(_ type: T.Type, from data: Data) throws -> T where T: Decodable {
        return try PropertyListDecoder().decode(type, from: data)
    }
}
```

- Single Responsibility Principle
- Open-Closed Principle
- **Liskov Substitution Principle**
- Interface Segregation Principle
- Dependency Inversion Principle

Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.



```swift
public protocol SectionMapper {
    /**
     Constructs a `SectionMapper` with a `SectionStorage` implementation.
     */
    init(_ storage: SectionStorage)

    /**
     Maps a `CodableDictionary`' content into a `Codable`.
     */
    func map(_ id: String, content: CodableDictionary?) throws -> Codable?
}
```


THANK YOU
STACKOVERFLOW
imgflip.com

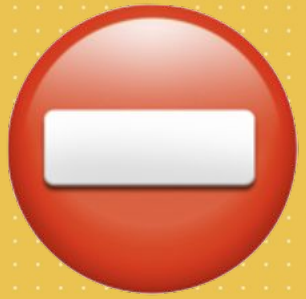
```
open class BaseSectionMapper<Model: Codable>: SectionMapper {
    // MARK: Public properties

    public let storage: SectionStorage

    // MARK: Init

    public required init(_ storage: SectionStorage) {
        self.storage = storage
    }

    /**
     Maps a `CodableDictionary`' content into a `Codable`.
     */
    open func map(_ id: String, content: CodableDictionary?) throws -> Codable? {
        guard let content = content else { return nil }

        do {
            let encoder = JSONEncoder()
            encoder.keyEncodingStrategy = .convertToSnakeCase
            let encodedContent = try encoder.encode(content)
            let decoder = JSONDecoder()
            decoder.keyDecodingStrategy = .convertFromSnakeCase
            return try decoder.decode(Model.self, from: encodedContent)
        } catch {
            throw SectionMapperError.mappingError(String(describing: error))
        }
    }
}
```

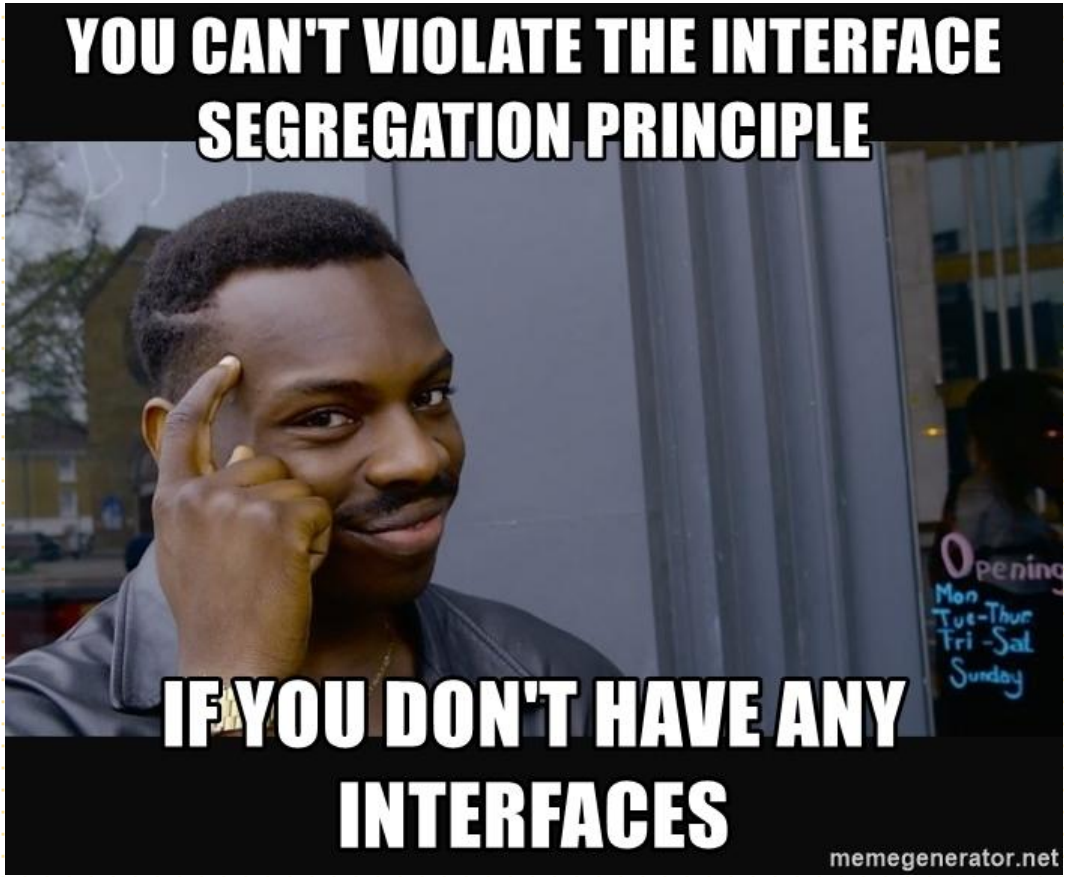
```
public class BankingMapper: BaseSectionMapper<BankingResponse> {
    public override func map(_ id: String,
                             content: CodableDictionary?) throws -> Codable? {
         let error = SectionMapperError.mappingError("There was an error decoding BankingResponse")

        do {
            guard let mappedContent = try super.map(id, content: content) as? BankingResponse else {
                throw error
            }

            // Custom banking logic

            return mappedContent
        } catch {
            throw error
        }
    }
}
```

```
let bankingMapper = BaseSectionMapper<BankingResponse>(storage)

let bankingCustomMapper = BankingMapper(storage)
```

- Single Responsibility Principle
- Open-Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Las clases no deben ser obligadas a implementar métodos de protocolos que no utilizan.




YOU CAN'T VIOLATE THE INTERFACE SEGREGATION PRINCIPLE
IF YOU DON'T HAVE ANY INTERFACES
Opening
Mon
Tue-Thur
Fri-Sat
Sunday
memegenerator.net

```swift
public protocol Api {
    func register(_ module: RegistryModule) throws
    func getMapper(for type: String) -> SectionMapper?
    func getProviders() -> [SectionProvider]
    func getProvider(for type: String) -> SectionProvider?
    func getStorage(for type: String) -> SectionStorage?
    func getListeners() -> [SectionListener]
}
```

```swift
/// This class manages all the home sections that want to be displayed.
public class SectionsApi: Api {
    /// The dictionary that contains all the modules with their respective `type`'s as keys
    private var modules = [String: RegistryModule]()

    public init() {}

    public func register(_ module: RegistryModule) throws {
        if modules[module.type] != nil {
            throw RegistryError.duplicatedKey("A Module with the type \(module.type) already exists")
        }
        modules[module.type] = module
    }

    public func getMapper(for type: String) -> SectionMapper? {
        return modules[type]?.mapper
    }

    public func getProviders() -> [SectionProvider] {
        return modules.map { $0.value.provider }
    }

    public func getProvider(for type: String) -> SectionProvider? {
        return modules[type]?.provider
    }

    public func getStorage(for type: String) -> SectionStorage? {
        return modules[type]?.storage
    }

    public func getListeners() -> [SectionListener] {
        return modules
            .compactMap { $0.value.listeners }
            .flatMap { $0 }
    }
}
```

```swift
/// This class manages all the home sections that want to be displayed.
public class SectionsApi {
    /// The dictionary that contains all the modules with their respective `type`'s as keys
    private var modules = [String: RegistryModule]()

    public init() {}

    /**
     - Returns: The current count of the registered `modules`.
     */
    public func getSectionCount() -> Int {
        return modules.count
    }
}
```

```
public protocol Registry {
    /**
     Registers a `RegistryModule` if wasn't registered previously.
     */
    func register(_ module: RegistryModule) throws
}
```

```
public protocol MapperRegistry {
    /**
     Obtains a `SectionMapper` for the given `type`.
     */
    func getMapper(for type: String) -> SectionMapper?
}
```

```
public protocol ProviderRegistry {
    /**
     Obtains all the registered providers.
     */
    func getProviders() -> [SectionProvider]

    /**
     Obtains a `SectionProvider` for the given `type`.
     */
    func getProvider(for type: String) -> SectionProvider?
}
```

```
public protocol StorageRegistry {
    /**
     Obtains a `SectionStorage` for the given `type`.
     */
    func getStorage(for type: String) -> SectionStorage?
}
```

```
public protocol ListenerRegistry {
    /**
     Obtains all the registered listeners.
     */
    func getListeners() -> [SectionListener]
}
```

```swift
extension SectionsApi: Registry {
    /**
     Registers a `RegistryModule` if wasn't registered previously.
     */
    public func register(_ module: RegistryModule) throws {
        if modules[module.type] != nil {
            let message = "A Module with the type \(module.type) already exists"
            throw RegistryError.duplicatedKey(message)
        }
        modules[module.type] = module
    }
}
```

```
extension SectionsApi: MapperRegistry {
    /**
     Obtains a `SectionMapper` for the given `type`.
     */
    public func getMapper(for type: String) -> SectionMapper? {
        return modules[type]?.mapper
    }
}
```

```swift
extension SectionsApi: ProviderRegistry {
    /**
     Obtains all the registered providers.
     */
    public func getProviders() -> [SectionProvider] {
        return modules.map { $0.value.provider }
    }

    /**
     Obtains a `SectionProvider` for the given `type`.
     */
    public func getProvider(for type: String) -> SectionProvider? {
        return modules[type]?.provider
    }
}
```

```
extension SectionsApi: StorageRegistry {
    /**
     Obtains a `SectionStorage` for the given `type`.
     */
    public func getStorage(for type: String) -> SectionStorage? {
        return modules[type]?.storage
    }
}
```

```
extension SectionsApi: ListenerRegistry {
    /**
     Obtains all the registered listeners.
     */
    public func getListeners() -> [SectionListener] {
        return modules
            .compactMap { $0.value.listeners }
            .flatMap { $0 }
    }
}
```

```
class MockListenerRegistry: ListenerRegistry {
    let listeners: [SectionListener]

    init(_ listeners: [SectionListener]) {
        self.listeners = listeners
    }

    func getListeners() -> [SectionListener] {
        return listeners
    }
}
```

- Single Responsibility Principle
- Open-Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- **D**ependency Inversion Principle

Los módulos de alto nivel no deben depender de los módulos de bajo nivel. Ambos deben depender de abstracciones.

```
public func homeView(_: HomeView, didSelect link: String) {
    guard let url = URL(string: link),
        let scheme = url.scheme else { return }

    if HWConfigurer.supportedDeeplinkSchemes.contains(scheme) {
        if let viewController =
            MLCommonRouter.sharedRouter().viewController(for: url, isPublic: false) {
            viewController.hidesBottomBarWhenPushed = true
            navigationController?.pushViewController(viewController, animated: true)
        }
    } else {
        if UIApplication.shared.canOpenURL(url) {
            UIApplication.shared.open(url, options: [:], completionHandler: nil)
        }
    }
}
```


I KNOW I'M SUPPOSED TO
CUT SOMETHING, BUT WHAT?
AND WHERE?

```swift
public func homeView(_: HomeView, didSelect link: String) {
    guard let url = URL(string: link),
        let scheme = url.scheme else { return }

    if HWConfigurer.supportedDeeplinkSchemes.contains(scheme) {
        if let viewController =
            MLCommonRouter.sharedRouter().viewController(for: url, isPublic: false) {
            viewController.hidesBottomBarWhenPushed = true
            navigationController?.pushViewController(viewController, animated: true)
        }
    } else {
        if UIApplication.shared.canOpenURL(url) {
            UIApplication.shared.open(url, options: [:], completionHandler: nil)
        }
    }
}
```

```
public protocol Navigator {
    /**
     Navigates to the `URL` target.
     */
    func navigate(with url: URL)
}
```

```swift
public protocol URLOpener {
    /**
     Checks if the url can be opened.
     */
    func canOpenURL(_ url: URL) -> Bool

    /**
     Opens the given url.
     */
    func open(_ url: URL, options: [String: Any],
              completionHandler completion: ((Bool) -> Swift.Void)?)
}
```

```
extension UIApplication: URLOpener {}
```

```swift
public class ExternalNavigator {
    // MARK: Private properties

    private let urlOpener: URLOpener

    // MARK: Init

    public init(_ urlOpener: URLOpener) {
        self.urlOpener = urlOpener
    }
}

// MARK: Navigator

extension ExternalNavigator: Navigator {
    public func navigate(with url: URL) {
        if urlOpener.canOpenURL(url) {
            urlOpener.open(url, options: [:], completionHandler: nil)
        }
    }
}
```

```
let externalNavigator = ExternalNavigator(UIApplication.shared)
```

```swift
public protocol Router {
    /**
     Builds a View Controller based on a given URL
     */
    func viewController(for url: URL, isPublic: Bool) -> UIViewController?
}
```

```
extension MLCommonRouter: Router {}
```

```swift
public class InternalNavigator {
    // MARK: Private properties

    private let router: Router
    private let navigationController: UINavigationController?

    // MARK: Init

    public init(_ router: Router, navigationController: UINavigationController?) {
        self.router = router
        self.navigationController = navigationController
    }
}

// MARK: Navigator

extension InternalNavigator: Navigator {
    public func navigate(with url: URL) {
        if let viewController = router.viewController(for: url, isPublic: false) {
            viewController.hidesBottomBarWhenPushed = true
            navigationController?.pushViewController(viewController, animated: true)
        }
    }
}
```

```swift
public class HomeNavigator {
    // MARK: Private properties

    private let internalNavigator: Navigator
    private let externalNavigator: Navigator
    private let supportedDeeplinkSchemes: [String]

    // MARK: Init

    public init(_ internalNavigator: Navigator, externalNavigator: Navigator,
                supportedDeeplinkSchemes: [String]) {
        self.internalNavigator = internalNavigator
        self.externalNavigator = externalNavigator
        self.supportedDeeplinkSchemes = supportedDeeplinkSchemes
    }
}
```

```
// MARK: Navigator

extension HomeNavigator: Navigator {
    public func navigate(with url: URL) {
        guard let scheme = url.scheme else { return }

        if supportedDeeplinkSchemes.contains(scheme) {
            internalNavigator.navigate(with: url)
        } else {
            externalNavigator.navigate(with: url)
        }
    }
}
```

Antes

```
public func homeView(_: HomeView, didSelect link: String) {
    guard let url = URL(string: link),
        let scheme = url.scheme else { return }

    if HWConfigurer.supportedDeeplinkSchemes.contains(scheme) {
        if let viewController =
            MLCommonRouter.sharedRouter().viewController(for: url, isPublic: false) {
            viewController.hidesBottomBarWhenPushed = true
            navigationController?.pushViewController(viewController, animated: true)
        }
    } else {
        if UIApplication.shared.canOpenURL(url) {
            UIApplication.shared.open(url, options: [:], completionHandler: nil)
        }
    }
}
```

Después

```
public func homeView(_: HomeView, didSelect link: String) {
    guard let url = URL(string: link) else { return }

    let navigator = resolver.resolveNavigator(with: navigationController)

    navigator.navigate(with: url)
}
```

Después

```
public func resolveNavigator(with navigationController: UINavigationController?) -> Navigator {
    let internalNavigator = InternalNavigator(MLCommonRouter.sharedRouter(),
                                              navigationController: navigationController)
    let externalNavigator = ExternalNavigator(UIApplication.shared)
    let schemes = HWConfigurer.supportedDeeplinkSchemes

    return HomeNavigator(internalNavigator,
                         externalNavigator: externalNavigator,
                         supportedDeeplinkSchemes: schemes)
}
```

```swift
@objcMembers public class BadgeBehaviour: MLBaseBehaviour {
    // MARK: Private properties

    private let badgeCleaner: BadgeCleaner
    private let notificationCenter: NotificationCenter

    // MARK: Init

    public init(_ badgeCleaner: BadgeCleaner, notificationCenter: NotificationCenter) {
        self.badgeCleaner = badgeCleaner
        self.notificationCenter = notificationCenter

        super.init()

        notificationCenter.addObserver(self, selector: #selector(cleanBadge),
                                       name: NSNotification.Name.UIApplicationDidBecomeActive,
                                       object: nil)
    }

    deinit {
        stopObserving()
    }

    public override func viewWillAppear(_: Bool) {
        cleanBadge()
    }

    public func cleanBadge() {
        badgeCleaner.clean()
    }

    public override func viewDealloc() {
        stopObserving()
    }

    private func stopObserving() {
        notificationCenter.removeObserver(self, name: NSNotification.Name.UIApplicationDidBecomeActive, object: nil)
    }
}
```

```
@objc public protocol BadgeCleaner: NSObjectProtocol {
    /**
     Cleans the badge of the app icon.
     */
    func clean()
}

extension UIApplication: BadgeCleaner {
    public func clean() {
        applicationIconBadgeNumber = 0
    }
}
```

```
- (NSArray <id <MLBehaviourProtocol> > *)requiredBehaviours
{
    BadgeBehaviour *badgeBehaviour = [[BadgeBehaviour alloc] init:[UIApplication sharedApplication]
                                                          notificationCenter:[NSNotificationCenter defaultCenter]];
    NotificationCenterBehaviour *notificationCenterBehaviour = [[NotificationCenterBehaviour alloc] init];

    return @[badgeBehaviour, notificationCenterBehaviour];
}
```

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Uncle Bob Martin https://en.wikipedia.org/wiki/Robert_C._Martin

Conclusiones

- Logran hacer que nuestro código sea más comprensible, flexible y mantenible.

- Aplicándolos, se obtiene una mejor encapsulación, una mayor cohesión, un bajo acoplamiento y un aislamiento más fuerte entre nuestras clases.

Preguntas

Gracias!

We're hiring 😉

joel.marquez@mercadolibre.com

@joelmarquez90

@joelmarquez

SwiftBA