

.....  
.....  
**iOS DESIGN  
PATTERNS**  
.....  
.....



**HANDS-ON CHALLENGES**

## iOS Design Patterns

Joshua Greene

Copyright ©2017 Razeware LLC.

### Notice of Rights

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written permission of the copyright owner.

### Notice of Liability

This challenge and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connection with the software or the use of other dealing in the software.

### Trademarks

All trademarks and registered trademarks appearing in this book are the property of their own respective owners.

# Table of Contents: Overview

Memento - Challenge .....	5
---------------------------	---

# Table of Contents: Extended

Memento - Challenge .....	5
Challenge.....	5
Challenge Solution .....	6

# Memento - Challenge

By Joshua Greene

Persisting the `BasicAuthToken` and `User` into `UserDefaults` works, but is it a good idea? Since `UserDefaults` doesn't use encryption, this isn't a great choice to be storing sensitive authentication and user data...!

The keychain is the right place to store this, yet it's more difficult to use. Writing a "keychain wrapper" class is beyond the scope of this challenge, but fortunately there are already several excellent, open-source options you can use!

## Challenge

Using your preferred dependency manager (e.g. CocoaPods, Carthage, etc), install **SwiftKeychainWrapper**. Here's the GitHub URL for this library:

<https://github.com/jrendel/SwiftKeychainWrapper>

Replace `UserDefaults` with `KeychainWrapper.standard` instead.

## Challenge Solution

This example solution uses CocoaPods to install **SwiftKeychainWrapper**. If you're new to CocoaPods and would like to learn how to use it, read our tutorial on it here:

<https://www.raywenderlich.com/97014>

Alternatively, feel free to use whichever dependency manager you prefer. ;]

Close **RWClean.xcodeproj** if you currently have it open.

Open terminal and cd into your working directory for your project.

Run `pod init` to create a default Podfile. Open it, and replace its contents with the following:

```
platform :ios, '10.0'
use_frameworks!

target 'RWClean' do
  pod 'SwiftKeychainWrapper', '~> 3.0'
end
```

Still in Terminal, enter `enter` and run `pod install` like normal to install this dependency.

Open the newly generated **RWClean.xcworkspace** and then open **AuthClient.swift**.

Add the following import to the top of the file:

```
import SwiftKeychainWrapper
```

Replace the **let persistantStore** declaration with the following:

```
fileprivate let persistantStore = KeychainWrapper.standard
```

Fortunately, KeychainWrapper has the exact same method signatures used by UserDefaults, so you don't need to make any other changes. Nice! ;]

**Build and run** and verify everything works as expected.

Note that you will need to sign in again, so your credentials will be stored in the keychain now.