SWIFT BEST PRACTICES

SINGLETONS – DIFFICULTY: EASY

INTRODUCTION

The singleton pattern restricts one type so that it can be instantiated only once. This type is then usually made globally accessible to the rest of your code using a shared instance, which makes it easy to share common functionality.

THE PROBLEM

Globally shared functionality causes problems with testing, concurrency, and more – while also making your code harder to understand, because a class can use functionality it wasn't explicitly given. Worse, it's easy to end up scattering references to the singleton throughout your code using code such as **Analytics.shared.post(...)**, so if you later change your mind about singletons you'll need to do a lot of rewriting.

THE SOLUTION

For dependencies under our control, one simple solution is to consider them implementation details and hide them away. This means your code still uses the singleton, but you don't actually reference the singleton directly. Using this approach allows you to move away from singletons easily later on, because all the singleton code is neatly centralized.

Here's a simple example singleton for us to work with:

```
class Analytics {
   static let shared = Analytics()

   private init() { }

   func post(_ notification: String) {
      print(message)
   }
}
```

Normally we'd use that by saying **Analytics.shared.post("Hello!")** or similar. If you want to hide that singleton away, you start with a protocol that defines the functionality of your singleton, like this:

```
protocol AnalyticsHandling {
    func post(_ notification: String)
}
```

Next you create an extension to that protocol that uses the singleton:

```
extension AnalyticsHandling {
    func post(_ notification: String) {
        Analytics.shared.post(notification)
    }
}
```

Now, anywhere you need to be able to post analytics messages you just add a conformance to the **AnalyticsHandling** protocol and you can call **post()**. This isolates your singleton to one protocol, making it easier to swap out if you change your mind later.

Note: for dependencies that aren't under our control, see the Dependency Injection document.

THE CHALLENGE

Our sample app use a Logger singleton, but references that singleton in various places. This means if we ever change our mind and want to use dependency injection or something else, we need to rewrite a lot of code.

Can you migrate the Logger singleton over to the protocol extension solution?