Learn the Four Swift Patterns I Swear By

Written by Bart Jacobs

Cocoacasts

Learn the Four Swift Patterns I Swear By

Bart Jacobs

This book is for sale at
http://leanpub.com/learn-the-four-swift-patterns-i-swear-by

This version was published on 2017-11-19

Contents

About Cocoacasts

My name is Bart Jacobs[1] and I run a mobile development company, Code Foundry[2]. I've been programming for more than fifteen years, focusing on Cocoa development soon after the introduction of the iPhone in 2007.

Over the years, I've taught thousands of people about Swift and Cocoa development. Through my experience teaching, I've discovered and learned about the main problems people struggle with.

[1] https://twitter.com/_bartjacobs
[2] https://codefoundry.be

I created Cocoacasts to offer a roadmap for anyone interested in learning Swift and Cocoa development. Through Cocoacasts, I provide a clear path to learn the tools, the language, and the frameworks you need to master Swift and Cocoa development.

I currently work as a freelance developer and teach people about Swift and Cocoa development. While I primarily focus on developing software for Apple's platforms, I consider myself a full stack developer with a love and interest for Swift and Ruby development.

You can find me on Twitter[3]. Follow me and say hi. You can also follow Cocoacasts[4] on Twitter if you're interested in what I teach on Cocoacasts.

[3] https://twitter.com/_bartjacobs
[4] https://twitter.com/_cocoacasts

Welcome

Swift is still very, very young and many developers are still figuring out how to best use the language. There are countless tutorials about patterns and best practices, which makes it hard to see the forest for the trees.

In this book, you learn the four patterns I use in every Swift project I work on. You learn how easy it is to integrate these patterns in any Swift project. I promise that they are easy to understand and implement.

1 Namespaces in Swift

The first Swift pattern I use in every project that has any complexity to it is namespacing with enums and structs. Swift modules make the need for type prefixes obsolete. In Objective-C, it's a best practice to use a type prefix to avoid naming collisions with other libraries and frameworks, including Apple's.

Even though modules are an important step forward, they're not as flexible as many of us would want them to be. Swift currently doesn't offer a solution to namespace types and constants within modules.

2 Dependency Injection in Swift

Dependency injection is a bit more daunting. Or that's what you're made to believe. Does dependency injection sound too complex or too fancy for your needs. The truth is that dependency injection is a fundamental pattern that's very easy to adopt.

My favorite quote about dependency injection is a quote by James Shore. It summarizes much of the confusion that surrounds dependency injection.

> Dependency Injection is a 25-dollar term for a 5-cent concept.
> â€” James Shore

When I first heard about dependency injection, I also figured it was a technique too advanced for my needs at that time. I could do without dependency injection, whatever it was.

3 Value Types and Reference Types

When talking about object-oriented programming, most of us intuitively think about classes. In Swift, however, things are a bit different. While you can continue to use classes, Swift has a few other tricks up its sleeve that can change the way you think about software development.

This is probably the most important mindset shift when working with Swift, especially if you're coming from a more traditional object-oriented programming language such as Ruby, Java, or Objective-C.

4 Model-View-ViewModel in Swift

Model-View-Controller, or MVC for short, is a widely used design pattern for architecting software applications. Cocoa applications are centered around MVC and many of Apple's frameworks are impregnated by the pattern.

But there's an alternative that resolves many of the issues MVC suffers from, **Model-View-ViewModel**. You've probably heard of MVVM. But why is it becoming increasingly popular in the Swift community? And why have other languages and frameworks embraced Model-View-ViewModel for years?

I hope that my book helps you in some way, big or small. If it does, then let me know. I'd love to hear from you.

Enjoy,

Bart

1 Namespaces

The first Swift pattern I use in every project that has any complexity to it is **namespacing** with enums and structs. Swift modules make the need for class prefixes obsolete. In Objective-C, it's a best practice to use a class prefix to avoid naming collisions with other libraries and frameworks, including Apple's.

Even though modules are an important step forward, they are not as flexible as many of us would want them to be. Swift currently doesn't offer a solution to namespace types and constants within modules.

A very common problem I run into when working with Swift is defining constants in such a way that they are easy to understand by anyone working on the project. In Objective-C, this would look something like this.

```
1 NSString * const CCAPIBaseURL = @"https://example.com/v1";
2 NSString * const CCAPIToken = @"sdfiug8186qf68qsdf18389qsh4niuy1";
```

This works fine, but it isn't pretty and easy to read. Even though Swift doesn't support namespaces within modules, there are several viable solutions to this problem.

Using Structures

The option I commonly adopt in Swift projects uses structures to create namespaces. The solution looks something like this.