SwiftUI for Absolute Beginners

Program Controls and Views for iPhone, iPad, and Mac Apps

Jayant Varma

Apress®

SwiftUI for Absolute Beginners

Program Controls and Views for iPhone, iPad, and Mac Apps

Jayant Varma

Apress®

SwiftUI for Absolute Beginners: Program Controls and Views for iPhone, iPad, and Mac Apps

Jayant Varma
Melbourne, VIC, Australia

ISBN-13 (pbk): 978-1-4842-5515-5   ISBN-13 (electronic): 978-1-4842-5516-2
https://doi.org/10.1007/978-1-4842-5516-2

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5515-5. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

This book is dedicated to my parents

Table of Contents

About the Author

Jayant Varma is a Developer, Consultant and author that has over 25 years of experience in developing and delivering applications, of which the last decade was focussed solely on iOS. He has worked on many iOS applications that span indie games to Enterprise level applications used by several users from the App store and via Enterprise builds. He loves to be hands on, being closer to the code and manages teams of iOS developers that work at different Enterprise level organisations in Melbourne, Australia. He has worked with/ made apps for NBN, Telstra, Westpac to name a few. He has written several books related to iOS development on topics like Lua, Swift, Objective-C, Xcode, Bash and now SwiftUI. His early days can be dated back to working with Z80 assembly, dBase, Clipper, FoxPro, Visual Basic and even RPG on AS/400.

He has experience in several domains. His love of code can see him dive deep hands-on into code. He is involved with the community and speaks at meetups and conferences, has worked in 3 major universities in Australia and teaches Swift to budding developers. He has also taught Swift to an Apple Education cohort.

He can be reached on Linked in at `https://www.linkedin.com/in/jayantvarma/`

About the Technical Reviewer

Mehul Mohan is an independent developer and security researcher who likes to work with code and create things with it. He runs codedamn (`https://www.youtube.com/codedamn`) as the platform to share his work with others, and also runs codedamn.com as an independent developer platform for learning and connecting. He's mostly into JavaScript and its runtimes but is eager to explore other interesting technologies. WWDC19 scholar, SwiftUI video series, and author of two books, you can find him using the handle @mehulmpt almost everywhere.

Acknowledgments

Writing a book is not exactly an easy task and more so when the technology is new and does not have a lot of information available. This has been a challenging journey also given the fact that every fortnight there were changes to the API that broke some of the earlier code or changed the way something was done. This book could be completed due to the support by my family, more so my lovely wife Monica who has supported me pulling late nighters and weekends trying out new features of SwiftUI and making sure that it all works and remains current.

A special thanks to my parents that always believed in me and to that effect this is going to be my 7th published book.

Thanks also to Aaron and Jessica at Apress, reaching out to Aaron for a new book pitch and get this process started was quite easy, he got things organised even while he was travelling and responded between his flights. Thanks to Jessica, who as usual made the process easy and a breeze and for all the quick responses and support at all aspects of the process.

Thanks to the Technical reviewer to go through the text and code and highlight the little changes that were missed and special thanks to the team that made this wonderful framework at Apple.

CHAPTER 1

What Is SwiftUI

In this chapter, we'll review the principles of SwiftUI and why it came into being. You'll see the advantages that it offers over traditional methodologies of development and how easy it is to write UI without having to worry much about it in a much more declarative manner.

The Beginnings

Every year, WWDC is always a source of exciting stuff for developers; everyone waits with baited breath to see what new tech is being introduced by Apple. Historically, WWDC, which is the Developers Conference, has been the forum where numerous new and interesting technologies have been previewed for release in some time. The most current groundbreaking piece of tech released by Apple was in 2014 when Apple released Swift, an alternative to the aging Objective-C. This not only had an easier syntax and was based on modern programming fundamentals but was also open sourced. Five years have passed since then and several books and apps are now created using Swift. This year in 2019, at WWDC, Apple released something that got developers all excited once again, called SwiftUI (Figure 1-1). Though this is still a new technology and at the time of writing the book still in beta, it can have some changes which would only add more functionality to the existing repertoire of SwiftUI.

J. Varma, *SwiftUI for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-5516-2_1

Figure 1-1. *Applications built using SwiftUI on Mac, iOS, and WatchOS*

In a single sentence, the easiest way to describe SwiftUI is a declarative UI. This might still not really answer or help understand what it is all about. An easy way to understand declarative UI is to simply state what you want, like "*I want my eggs hard boiled*" instead of detailing the steps like "*get eggs, put them in a pan filled with water, put this on the flame and wait for 7 minutes.*" Its more about focusing on what's important than how to achieve that.

In traditional programming languages, one would generally create an UI element; then set its visual frame; set the colors, background and foreground, and other attributes; and then set it up on the visual hierarchy. With declarative, you only need to specify that you need an element, and these can then be modified using modifiers.

SwiftUI Principles

SwiftUI is built on four principles discussed in the following sections.

Declarative

Traditional development was focused more on how to create elements and how to display them on the screen and then continue to update them as the data changes. With a declarative UI, this moves away from that, instead allowing the developer to focus on what you want to display.

Let's see how this looks currently:

```
let labelText = UILabel(frame: CGRect(x:0, y:0, width:100,
height:100))
labelText.text = "Hello World"
labelText.textColor = UIColor.blue
labelText.backgroundColor = UIColor.red
labelText.font = UIFont(name: "Helvetica", size: 24)
self.view.addSubview(labelText)
```

This simply creates a `UILabel` and then sets its attributes. This code is specific for iOS as it uses the `UILabel` which is not available on macOS which uses `NSLabel` or the `watchOS` which uses `WKInterfaceLabel`. Now with SwiftUI, there is a common element that is available on all of `iOS`, `iPadOS`, `macOS`, and `watchOS`. The same code looks like

```
    Text("Hello World")
    .color(.blue)
    .background(Color.red)
    .font(.largeTitle)
```

That brings us up to the next principle:

Automatic

This principle is hinged on the basis that it offers automatic functionality; if you saw the preceding code snippet, there was no mention of spacing, frames, insets, and the like. SwiftUI offers all the out-of-the-box features for free functionality - like Localization; if the code had language strings, then the line above would display the localized version, all without writing extra code, all automatically. Developers can also take advantage of functionality like left-to-right, Dark mode, Dynamic type, and more, all with writing minimal code.

Composition

This is another interesting principle that SwiftUI is based on simply because a UI is nothing but a collection of visual elements that together provide the user an interactive experience. With SwiftUI, Apple makes this much easier to manage, even creating complex views by using containers like `VStack` or `HStack`. Composition is nothing but creating newer elements by compositing using other elements.

Consistent

Now when Apple wanted to create an easy-to-use application program interface (API), they made sure that it was easy to use. The biggest problem faced with developers is updating the UI from data models; there can be lags and/or issues that prevent the data from being used in the update cycle and can lead to strange errors or behaviors that are difficult to understand. So, to solve this particular problem with data and UI, the fourth principle is important.

Since the UI is a reflection of the data it represents, it should always be in sync so as to provide a consistent experience. Traditionally, this is the step that is error prone as data can be out of sync and/or updated out of cycle. With SwiftUI, the UI updates automatically as soon as the data changes.

It also caters for a temporary UI state that can be simply declared using the `@State` property wrapper.

Note With traditional programming, most developers are used to mutability; with SwiftUI, it is surprising how little mutability is required.

SwiftUI Architecture

The advantages of using SwiftUI are not limited to the preceding points; these are just the tip of the iceberg, mostly because it is not very long ago that SwiftUI was released and like the early versions of Swift, there are a lot of changes to be expected. However, most of the principles would still be useful and available even with the organic changes.

The Swift language is open source and there is an evolution web site where the community discusses and progresses the development. SwiftUI is however not open source and managed only by Apple. It is cross platform on the Apple Ecosystems only and works across all of them, iOS, iPadOS, macOS, tvOS, and watchOS (Figure 1-2).

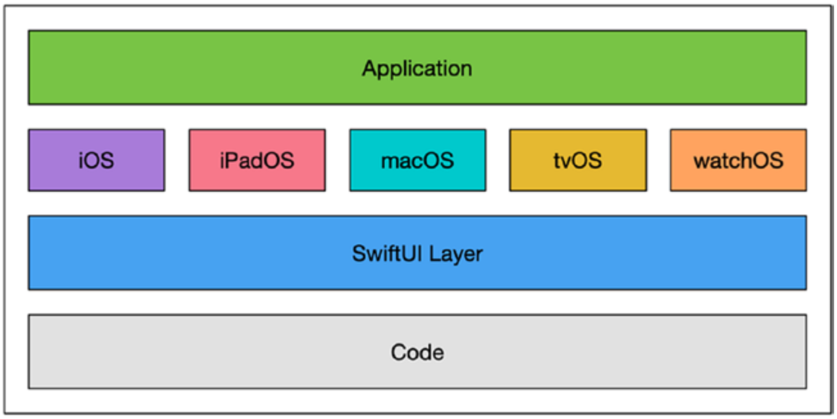

Figure 1-2. *SwiftUI Architecture*

While SwiftUI sits on top of the code and creates the application that displays UI elements, it does not, and please note this (as of now and probably even later), it does not create native elements from the code. So when a developer creates a `text` element, it does not create a `UILabel` or a `NSLabel` or `WKInterfaceLabel`. It is still a `text` element, and in the view debugger, it shows all of the elements, the native ones and the SwiftUI. However, they are displayed separately in their own hierarchies. All of the SwiftUI is hosted in a container called Hosting View; more details of all these are available in subsequent chapters.

Requirements to Use SwiftUI

There are a couple of touch points that use SwiftUI, the first being the newer OS, iOS 13, iPadOS 13, macOS 15, and watchOS 6. From a development perspective, the minimum requirements are Xcode 11 or higher running on macOS 10.15 Catalina or higher, and from a language perspective, it needs the new features added in Swift 5.1. With all of these, it is apparently clear that it is not available with Objective-C; perhaps, the key giveaway was the name SwiftUI and not a name that was generic.

Integration with Xcode

The second advantage that SwiftUI offers after it being an easy declarative UI language is that it offers quick previews. With Xcode 6, Apple offered a `@IBDesignable` attribute that allowed developers to create classes that could be previewed in Interface Builder and interactively change some parameters and see the changes accordingly. SwiftUI allows us to create views and also provides sample data to preview the view in Xcode without having to run it. Xcode compiles the code and displays the preview all in the background as soon as some code is written. If there is a substantial change, then previews are paused, and requesting to resume the preview would compile the code and attempt to preview the UI.