Make the leap from front-end web development to iOS development with Swift

Migrating to Swift from Web Development

Sean Liao | Mark Punak

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

In 2000, I created my first mobile app for an inventory-tracking project using PalmOS handheld devices. The initial project was a full-staffed team effort that consisted of mobile developers, SAP consultants, supply-chain subject-matter experts, middleware developers, QA testers, architects, business sponsors, and so forth. JavaME came up strong in 2002, followed by Pocket PC/Windows Mobile. I did several mobile projects in which I converted mobile apps to the Pocket PC platform by blindly translating C++ to JavaME to C# .NETCF mobile code. These "translation" efforts prolonged the whole product life cycle. The project achieved a higher return on investment (ROI) by extending the product life because the extra cost of translating mobile code was surprisingly low. Ever since then, I have been translating front-end mobile apps among various mobile platforms. In recent years, most of my work has involved porting mobile apps between Android and iOS and mobilizing existing web sites. Porting apps between iOS and Android is fairly straightforward. This is also true for porting mobile web apps using a RESTful service. Even for traditional non-service-oriented web apps, you still want to follow the same path: reusing existing business cases and software artifacts and reaching a bigger audience to maximize the ROI.

One thing is for sure: there are a lot demands for mobilizing existing web apps to reach mobile users. That's why I decided to write this book.

The primary objective of this book is to help experienced web developers leap into native iOS–Swift mobile development. It is easier than you think, and this book will make it even easier with step-by-step guidelines. You can immediately translate common mobile use cases to iOS.

Who Is This Book For?

This book was specifically written for web developers who want to make iOS mobile apps. The book will show you the common iOS programming subjects and frameworks by relating them to your familiar web programming tasks when appropriate.

How This Book Is Organized

In part I, you will get the iOS Xcode integrated development environment (IDE) up and running. You will be guided in creating tutorial projects that will become your porting sample projects. I believe this is the best way for you to get hands-on experience while learning programming topics.

Part II of this book shows you how to plan and structure your iOS apps by creating a storyboard and breaking the app into model-view-controller (MVC) classes. The common mobile topics are followed, including creating a user interface, managing data, and enabling networking with remote services. You will then be able to create simple but meaningful iOS apps with rich UI components and be able to handle common create, read, update, delete (CRUD) operations locally and remotely.

Last, part III walks you through a case study for a complete iOS app. It recaps the topics in this book. You can also use the book's table of contents or index to help find the mobile topics you need.

A bonus chapter was added in the end reveals how to mix and match web front development with iOS SDK, the so-called hybrid apps. You may choose to bundle the web contents and HTML pages with Javascript code just like you normally do for frontend web apps. You can interface with the native iOS platform features and communicate between your JavaScript and iOS code back and forth.

When you complete this journey, you will be able to use Xcode and Swift to effectively implement simple and meaningful iOS apps.

Part 1

Prepare Your Tools

Chapter 1

Setting Up the Development Environment

It is more fun to see apps run than to read the source code, and you cannot get hands-on programming experience by just reading books. Let's get the development environment up and running first so you can use it—and learn Swift programming for iOS along the way.

There is no single integrated development environment (IDE) that can be called *the* IDE for web development. Eclipse and Eclipse-based products such as Aptana and Spket, and others such as NetBeans, IntelliJ Idea, and Visual Studio, are all popular tools for building and deploying web applications. In fact, some outstanding web developers use only a text editor to create Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), or JavaScript files! In the iOS programming world, Apple purposely requires a single development environment for creating iOS apps. It makes for no jailbreakers, and all you need is the one tool: Xcode.

> **Note** Rather than attempt to provide specific examples from every web IDE, this book will instead reference analogous tasks from web development where applicable.

Xcode and the iOS SDK

Xcode is a complete tool set for building iOS apps. In other words, it is an IDE that helps you build, test, debug, and package your iOS apps. It is free, but you must have an Intel-based Mac running Mac OS X Mavericks or newer. You will use the latest Xcode, version 6, throughout this book.

Installing from the Mac App Store

Xcode is distributed via the Mac App Store, which takes care of the download and install for you. With a single click to start the download and installation of Xcode, you get the compilers, code editor, iOS software development kit (SDK), debugger, device emulators, and everything you need to create iOS apps. Figure 1-1 shows Xcode in the Mac App Store.

***Figure 1-1.** Xcode in Mac App Store*

All you need to do is install the latest Xcode from the App Store. After completing the installation, launch Xcode from the Applications folder. Keep it in the Mac OS Dock so that you can launch it at any time.

The first time you launch Xcode, it immediately prompts you to install the required components (see Figure 1-2). Click Install to complete the Xcode installation.

Figure 1-2. Install the required components

After the required components are installed, you should see the screen in Figure 1-3. Your iOS IDE, Xcode, is ready!

Figure 1-3. Welcome to Xcode screen

Creating an iOS Project Using the Template

WEB ANALOGY

You can create a New Web Application project in Visual Studio, create a new project in Sencha Architect (and select a framework), use the web app project templates in Eclipse web projects, and so on.

You've got the right tool; now, wouldn't you like to see some real action—like creating an iOS app and seeing it run? I'd like that, too! You want to do this to ensure your IDE is working properly as well.

Out of the box, Xcode offers the project creation templates that immediately give you a starting point that contains the minimal software artifacts for the given project types. The objective of this section is to show you how to create an iOS app as quickly as possible. Hold any programming questions so you can finish the project as fast as you can. For now, complete the following steps:

1. Launch Xcode if you haven't launched it yet.
2. Click "Create a new Xcode project" on the Welcome to Xcode screen (shown earlier in Figure 1-3). Figure 1-4 shows the prompt that asks you to choose a template for your project.
    a. In the left panel of Figure 1-4, select iOS ➤ Application.
    b. In the right panel of Figure 1-4, you may choose any of the templates. Just for fun, choose Game.

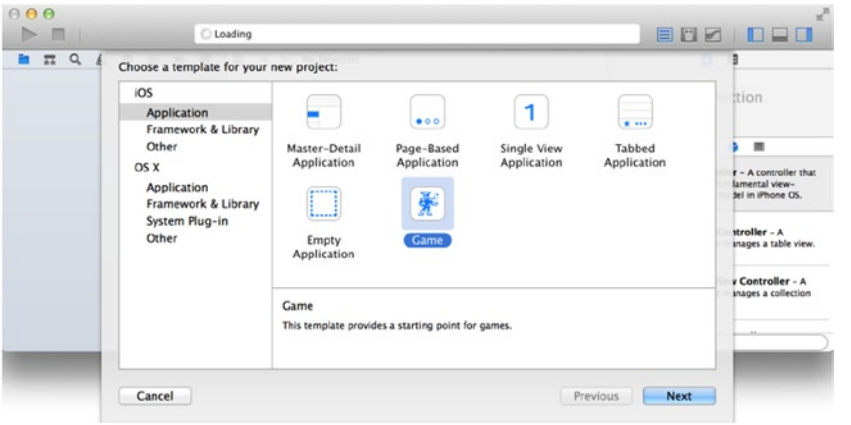

Figure 1-4. *Choose a template*

3. Click the Next button.
4. Figure 1-5 depicts the basic project information that requires you to fill in the following:
    a. *Product Name*: This is the app name. Name your project LessonOne.
    b. *Organization Name*: This is optional; for example, you can use your organization's name or any name you choose.

c. *Organization Identifier*: Together with the product name, the organization identifier should uniquely identify your app. A reverse domain name is recommended (for example, `com.yourdomain.xxx`).

d. *Language*, *Game Technology*, and *Devices*: You don't need to change these settings.

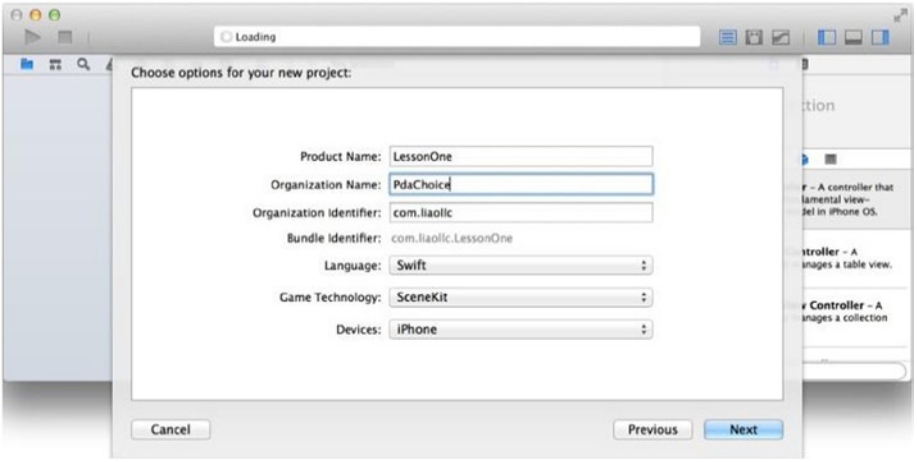

Figure 1-5. *iOS project options*

5. Click the Next button when done.

6. Select a folder in which to save your LessonOne project.

That is it! You just created an iOS project, the LessonOne project, that you can see the in Xcode as shown in Figure 1-6.

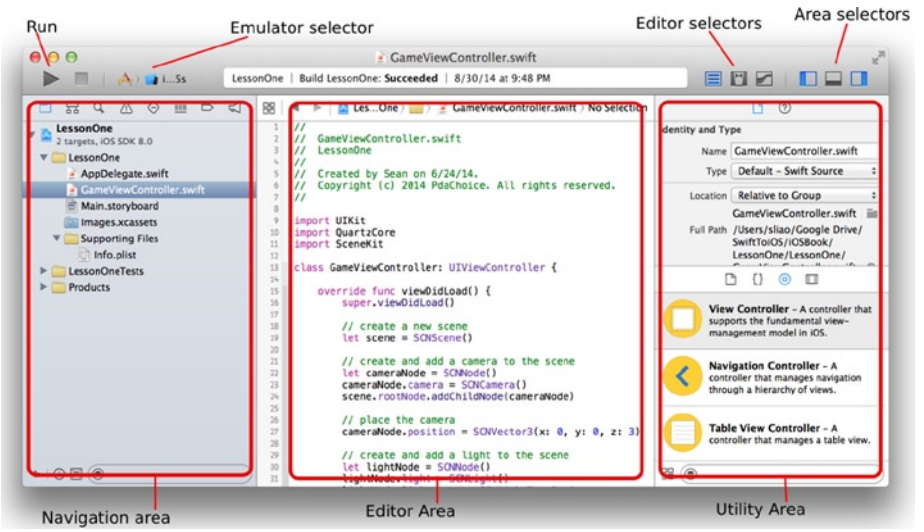

Figure 1-6. LessonOne project in Xcode Project Navigator (left panel)

The LessonOne project can be seen on the left panel, as shown in Figure 1-6; this is the Project Navigator in the navigation area. The Xcode project template creates the project folder, the application source code, and all the resources for building the LessonOne apps.

Building the Project

WEB ANALOGY

This is the process that bundles, minifies, and packages your app and all the referenced code that will be deployed with it.

To build and compile the Xcode project, use the Build action, which is located in the Product ➤ Build menu in Xcode (or press ⌘+B). You will get used to using the ⌘+B keyboard shortcut a lot because Xcode doesn't automatically build your code (unlike Eclipse ADT, which builds it automatically by default).

Launching the App

The LessonOne project should have no errors. You can launch the app and see it run on an iOS emulator. The emulator is an important piece of any IDE, probably even more important for mobile apps. All the iOS devices

emulators are right there in Xcode, and you can launch the LessonOne project on the selected device, including the iOS emulator, by clicking the triangle button in the upper-left corner, as shown earlier in Figure 1-6.

Alternatively, you can use the ⌘+R Xcode keyboard shortcut key for the Run action to launch the app. You should see your LessonOne app running on the iPhone emulator, as shown in Figure 1-7.

Figure 1-7. *LessonOne app in the emulator*

Play with the app and then select other emulators from the device drop-down selector (see the pointer in Figure 1-6). A mouse-click event on an emulator is equivalent to a touch event, and three-finger movement on the trackpad is equivalent to a touch-drag on a physical iOS screen. If you don't have a particular device yet, definitely play with the emulator to get familiar with the emulated iOS devices.

> **Tip** To change to landscape or portrait orientation, press ⌘+left arrow or ⌘+right arrow to rotate the emulator.

The iOS emulators are robust and responsive, and they behave almost exactly like real devices. For learning Swift programming for iOS, the emulator actually is better. In this book, you are not required to run apps on a physical iOS device; for that you would need to be a registered iOS developer. You can save the $99 iOS developer membership fee until you are ready to submit your first app to the App Store or for when your app requires certain features not available in the emulator (for example, the camera or certain sensors). For your convenience, I have provided how

to build and sign an app for on-device debugging in the appendix of this book. For now, if your app is launched and running on an iOS emulator, your mission is completed!

Summary

By installing Xcode 6, you immediately have a fully functional IDE ready to create iOS apps without hassle. This chapter walked you through the basic project creation tasks in Xcode 6, using an iOS project template to start your first iOS project. This chapter also showed you how to build and run your iOS app in iOS emulators. You haven't written any code yet, but your Xcode tool is working and verified. You will learn more and gain hands-on programming experience from the guided exercises in the following chapters.

Chapter 2

iOS Programming Basics

Creating mobile apps for both iOS and web deployment is fun and rewarding. With Xcode in place, you are ready to write code, build, and run iOS apps now. Objective-C had been the primary programming language for iOS apps until Swift was officially announced at the 2014 Apple Worldwide Developers Conference. If you're just starting to learn iOS programming, you should go with Swift because there is no reason to choose the old way and miss the latest and greatest features. Your next steps should be learning the fundamentals of the following:

- The Swift programming language
- The anatomy of the iOS project and the Xcode storyboard editor

The purpose of this chapter is to get you comfortable with reading the Swift code in this book. To achieve this goal, you will be creating a HelloSwift project while learning about Swift programming language highlights.

You will create another Xcode iOS project in the second part of the chapter. All iOS apps have a user interface (UI). You normally start by creating the UI using the most important Xcode tool, the storyboard editor, which draws the UI widgets and components and connects them to your code. You also will see the typical iOS project structures and components while creating this iOS app. You may not need to understand everything about the iOS framework in the beginning, but the first storyboard lesson should be "just enough" for you to get a feel for the different programming paradigm. Later, the materials in Chapters 3 and 4 continue with step-by-step instructions for common programming tasks and framework topics. Follow these mapping instructions, and the ideas will more easily stick with you as you get a broader picture of the whole app.

The Swift Language in a Nutshell

Swift is the newest programming language for creating iOS apps. I am confident that learning the Swift language won't be the highest hurdle for you; JavaScript and C# developers will pick up Swift code naturally because they are syntactically similar and follow the conventions of typical object-oriented (OO) programming languages. Just to give you a quick preview, Table 2-1 briefly compares JavaScript to Swift.

***Table 2-1.** Java-to-Swift Language Syntax Comparison in a Nutshell*

JavaScript	Swift
`<script type="text/javascript" src="scripts/packagename/Xyz.js"> </script>`	`import framework`
`Xyz.prototype = new SomeClass();`	`class Xyz : SomeClass`
`var mProperty;`	`var mProperty : Int`
`Xyz.prototype.constructor=Xyz // constructor`	`init()`
`var obj = new Xyz();`	`var obj : Xyz = Xyz()`
`Xyz.prototype.doWork(arg)`	`func doWork (arg: String) -> Void`
`obj.doWork(arg);`	`obj.doWork(arg)`

The Table 2-1 cross-reference refers to how native JavaScript prototype-based inheritance might be used to implement classical inheritance used by Swift, Java, and C#. For those familiar with Prototype and Sencha, the extension libraries and methods for inheritance should make the transition even more straightforward. If you come from a basic JQuery programming background, the syntax of Swift will seem familiar, but the structure of your code will move from a flat list of functions to encapsulated objects and object instances with a hierarchal source structure. In this section, I will not discuss the in-depth OO theory or techniques. However, I do want to point out certain important ideas for pure-JavaScript developers.

HelloSwift with Xcode

Instead of my describing the uses and syntax rules in a formal way, you are going to create a HelloSwift Xcode project and write the code listing from Table 2-1 yourself. You will also perform the following common Xcode programming tasks: creating a class, building and running a project, and using the debugger.

Creating a Swift Command-Line Project

Let's create a command-line Swift program because it is really simple and you can focus on the Swift language without being sidetracked by other questions.

Follow these instructions to proceed:

1. Launch Xcode 6 if it is not running. You should see the Welcome to Xcode screen. Click "Create a new Xcode project." Alternatively, you can select File ➤ Project from the Xcode menu bar.
2. Choose OS X and then Application in the left column and then select the Command Line Tool template in the "Choose a template for your new project" window (see Figure 2-1).

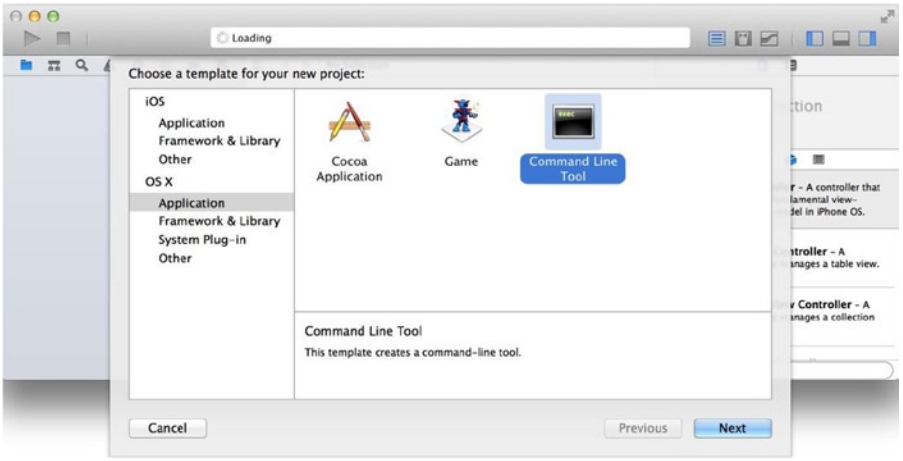

Figure 2-1. *Choosing an Xcode template*

3. To finish creating the new project with the template, follow the same onscreen instructions that you used to create the LessonOne project in Chapter 1:
   a. *Product Name*: Enter **HelloSwift**.
   b. *Organization Name*: You can use anything here, such as PdaChoice.
   c. *Organization Identifier*: You can use anything here, such as `com.liaollc`.
   d. Language: Select Swift.

4. Click the Next button when done.
5. Select a folder in which to save your HelloSwift project.
6. The HelloSwift project appears in the Project Navigator (see Figure 2-2).

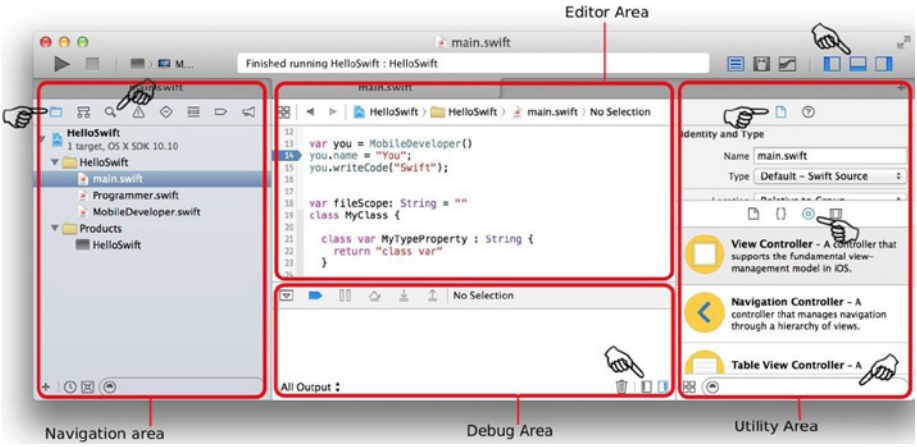

***Figure 2-2.** Creating the HelloSwift project*

The command-line template creates the `main.swift` file for you. This is the entry point of the OS X command line tool program. You will be writing code in `main.swift` to demonstrate common object-oriented code.

Figure 2-2 shows that the typical Xcode workspace contains three areas from left to right and a top toolbar. Inside each area, there are subviews that you may switch to using the selector bars.

- The Project Navigator area is on the left. Similar to many IDE, this is where you can see the whole project structure and select the file you want to edit. There are other views in this area; for example, you can enable Search view by selecting the Search icon in the selector bar.
- The Editor area in the middle shows the selected file in its editor, in which you can edit the file, write your code, or modify project settings depending on the file selected. The Console and Variable views are inside the Debug area. Most likely you will want to show them during debugging sessions. You can hide or show them by clicking the toggle buttons on the top and bottom toolbars.

- The Utility area on the right contains several inspector views that allow you to edit attributes of the whole file or the item selected in the Editor area. Depending on the type of files you select, different types of inspectors will be available in the top selector bar. For example, you will have more inspectors showing in the selector bar if you are editing a screen or UI widgets. The bottom of the area is called the Libraries area. Use the selector bar to select one of the library views. You can drag and drop items from Libraries to the appropriate editor to visually modify file content. You will use the Object Library a lot to compose UIs visually.

Click any of the icons on the selector bars, or hover your mouse over the pointer in Figure 2-2, to see the hover text tips in the workspace to get yourself familiar with Xcode workspace. The subviews appear more condensed than those in most of the IDE I ever experienced, but essentially it is a tool for the same purpose: editing project files and compiling, building, debugging, and running the executables. You will use it repeatedly throughout the book.

Creating a Swift Class

A class is the fundamental building block of any classic OO programming language. A class is a software template that defines what the objects know (state) and how the objects behave (methods). JavaScript uses functions and their prototypes to accomplish the same goal.

While JavaScript web applications often make use of objects by creating and manipulating DOM elements, it is entirely possible to create a JavaScript web application without creating a single new reusable object structure. Using anonymous objects and scripts running in the global scope, referencing function libraries like JQuery, is quite common. Swift, however, requires developers to structure their code into named class types before the objects can be created.

To create a new Swift class, you can create it in the existing `main.swift` file, or you can follow the Java convention to create it in its own file as shown in the following steps:

1. Expand the newly created HelloSwift project, right-click the `HelloSwift` folder to bring up the folder context menu (see Figure 2-3), and select New File.

    a. Select iOS and then Source from the left panel and then select the Swift File template in the “Choose a template for your new file” screen.

    b. Save the file and name it `MobileDeveloper.swift`. The file should appear in your project.

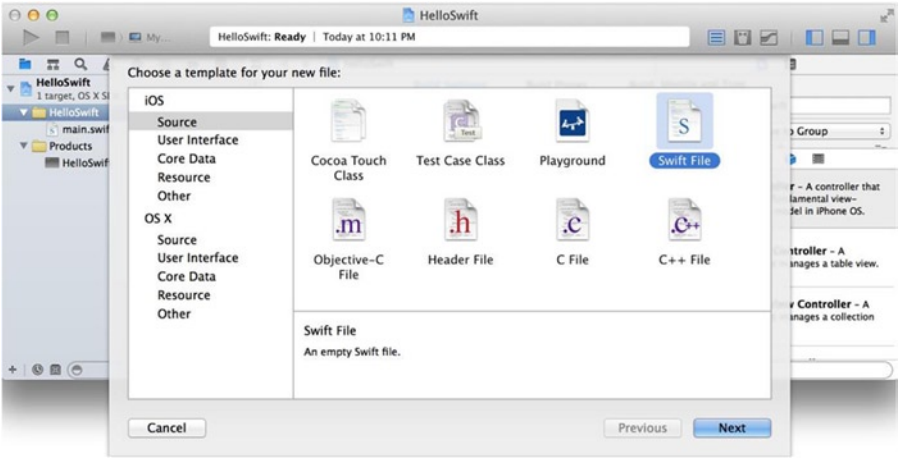

Figure 2-3. Create a Swift class from the folder context menu

2. Enter the code in Listing 2-1 in the `MobileDeveloper.swift` file to create the `MobileDeveloper` Swift class.

> **Note** Unlike Java, a Swift class doesn't implicitly inherit from any class. It can be the base class on its own.

Listing 2-1. Declare MobileDeveloper Class

```
class MobileDeveloper {

}
```

3. Create a property called `name` by declaring a variable inside the class (see Listing 2-2). This is called a *stored property* in Swift, where the variable type is inferred by the assigned value (known as *type inference* in Swift).

Listing 2-2. Stored Property in Swift

```
class MobileDeveloper {
   var name  = "" // var type, String, is inferred by the value
}
```

> **Note** The semicolon (`;`) is optional for terminating a statement in the same line.

Creating a Swift Protocol

WEB ANALOGY

Though JavaScript has no native comparable for a Swift protocol, web developers using C# or PHP for their server-side code will find comparable examples in the use of interfaces.

In object-oriented programming (OOP), an interface or protocol is essentially a predefined set of behaviors, methods, or properties. The protocol provides no implementation of the methods themselves but rather defines the method names, parameters, and return types. Consumers of the protocol can count on objects they are consuming to properly implement the items defined in the protocol. Protocols allow objects of different types and class inheritance to provide a common, strongly typed interface, thus implementing the concept of polymorphism, one of the fundamental concepts of OOP. You may also provide multiple implementation classes for the same contract and programmatically supply the appropriate instances in the runtime.

In simpler terms, this is a great way to explicitly break dependencies between callers and callees because callers and callees can be implemented independently with clearly defined programming contracts, called *protocols* in Swift.

Create a Swift protocol called `Programmer` by doing the following:

1. Right-click the `HelloSwift` folder to create the `Programmer.swift` file.
2. In the Editor area, create the `Programmer` protocol with the method `writeCode(...)`, as shown in Listing 2-3.

Listing 2-3. Declare the Programmer Protocol

```
protocol Programmer {
  func writeCode(arg: String) -> Void
}
```

Implementing the Protocol

To conform to the expected behavior defined in a Swift protocol, the tagged class must implement the methods defined in the protocol. To make the `MobileDeveloper` class implement the `Programmer` protocol, do the following:

1. Modify `MobileDeveloper.swift` and declare the `MobileDeveloper` class to implement the `Programmer` protocol, as shown in Listing 2-4.

   Listing 2-4. Conform to MobileDeveloper Protocol

   ```
   class MobileDeveloper : Programmer {
     ...
   }
   ```

> **Note** If the Swift class already has a superclass, list the superclass name before any protocols it adopts, followed by a comma (`,`)—for example:
>
> ```
> class MobileDeveloper : Person, Programmer
> ```

2. Provide the `writeCode(...)` method implementation body, as shown in Listing 2-5.

   Listing 2-5. Method Body

   ```
   class MobileDeveloper: Programmer {
     ...
     func writeCode(arg: String) -> Void {
       println("\(self.name) wrote: Hello, \(arg)")
     }
   }
   ```

> **Note** `\(self.name)` is evaluated first inside the quoted `String` literal.

Using the Swift Instance

WEB ANALOGY

```
var you = new MobileDeveloper();
you.setName("You");
you.writeCode("Javascript");
```

You have created a Swift `MobileDeveloper` class and implemented the `Programmer` obligations in pretty much the same way you would in Java with some minor syntax differences. To use the class, it is the same as Java in principle, calling a method defined in the receiver from the sender. Modify `HelloSwift/main.swift` as shown in Listing 2-6.

Listing 2-6. Swift Entry main.swift

```
var you = MobileDeveloper()
you.name = "You"
you.writeCode("Java")
```

Implementing Access Control

WEB ANALOGY

While JavaScript doesn't have access control keywords, many JavaScript developers use closures and constructor variables to create methods and variables with private access from the object.

Here's an example:

```
function MyObject() {
    var private;
    this.getPrivate = function (arg) {
        return private;
    }
```

```
    this.setPrivate = function (val) {
       private=val;
    }
}

var test =new MyObject();
test.setVal("My private value");
console.log("Private value is "+test.getVal());
```

Encapsulation is one of the fundamental principles of OOP; in a nutshell, certain internal states or methods are meant only for internal implementations but not to be used directly by the callers. Swift provides access controllers to prevent access to the members or methods that developers decide to hide. This is achieved with access modifiers for files and with module access controls using the following keywords as the access modifiers: `private`, `public`, and `internal`. The `internal` access modifier is the default access control that is public to the whole module but not visible when the modules are imported. If you are building a reusable module that can be imported by other program, use the `public` access modifier to expose the API to another module. The `private` access modifier makes your custom classes or type or members visible only within the file scope.

To demonstrate the `private` access control, modify `MobileDeveloper.swift` as shown in Listing 2-7.

Listing 2-7. The private Access Modifier in MobileDeveloper

```
private class MobileDeveloper {
  var name  = "" // var type is infered by the value

  func writeCode(arg: String) -> Void {
    // some dummy implementation
    println("\(self.name) wrote: Hello, \(arg)")
  }

  private func doPrivateWork() {
    println(">> doPrivateWork")
  }
}

// another class in the same source file
class TestDriver {
  func testDoPrivateWork() -> Void {
    var developer = MobileDeveloper()
    developer.doPrivateWork()
  }
}
```

Since `TestDriver` is implemented in the same source file, the code in `TestDriver` still can access the private class and its private method. However, the `MobileDeveloper` class in Listing 2-6 becomes not visible anymore from the `main.swift` file.

Using the Xcode Debugger

Knowing how to use the debugger when creating software can make a big difference in your productivity. Do the following to see the common debugging tasks in the Xcode debugger:

1. To set a breakpoint, click the line number in the Xcode code editor. Figure 2-4 depicts a breakpoint that was set in the `main.swift` file.

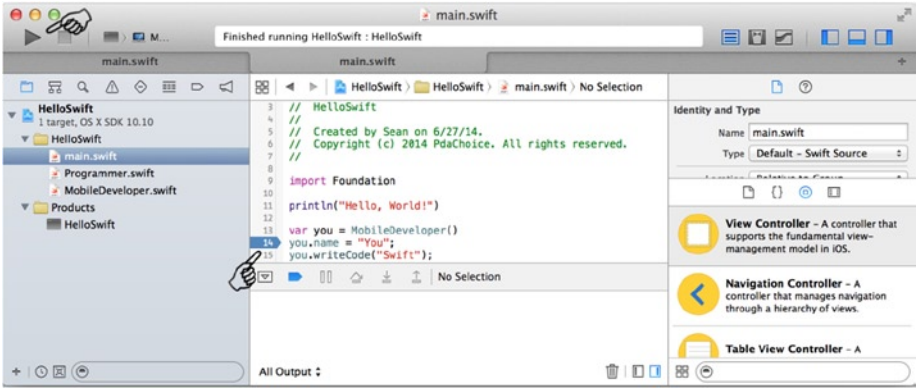

Figure 2-4. *Breakpoint*

> **Note** To turn on line numbers in Xcode editors, go to the Xcode top menu bar and select Xcode ➤ Preferences ➤ Text Editing ➤ Show Line Numbers. There are other handy settings there that you may want to look at (for example, shortcut keys are defined under Key Binding).

2. To run the HelloSwift project, click the triangle-shaped Run button in the upper-left corner or press ⌘+R (see Figure 2-5).