Migrating to Android for iOS Developers

Sean Liao

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 started my first PalmOS mobile app for an inventory-tracking project. The initial project was a full-staffed team effort that consisted of mobile developers, SAP consultants, supply-chain SME, J2EE middleware developers, QA testers, solution architects, business sponsors, and so forth. JavaME came up strong in 2002, followed by Pocket/Windows Mobile. I did several mobile projects converting the mobile apps to the PocketPC platform by blindly translating JavaME mobile code to C# .NETCF mobile code. The "translation" efforts prolonged the whole product life cycle. The project achieved higher ROI as the product life extended, because the extra cost of translating mobile code was surprisingly low. Ever since then, I have been translating front-end mobile apps among JavaME, BlackBerry, and Windows Mobile platforms.

In 2009, by repeating the same porting process, I created my first simple iOS app by translating a Windows mobile app. That started my iOS programming journey and which eventually led me to becoming a fulltime iOS developer. It was a no-brainer for me to try porting to Android later.

When you have the whole solution completed for your iOS app, all the issues have been verified and the other deliverables and project artifacts are already reusable. Knowing the Android market share, I always clone my iOS apps to Android. The return on investment (ROI) immediately gets improved because the level of efforts for the Android porting proves to be only a fraction of the entire project's effort—again and again. It would be just a waste to not do it.

The primary objective of this book is to help experienced iOS developers leap into native Android mobile development. It is easier than you think, and this book will make it even easier with iOS analogies (and mapping guidelines) so you can immediately translate common mobile use cases to Android.

Who Is This Book For?

This book is specifically written for iOS developers who want to take advantage of their mobile knowledge and make the mobile applications available on the Android mobile platform. The book will show you that you already have the fundamentals for the Android platform. Let me show you that you are very close to becoming an Android developer. Let me show you the common programming subjects and frameworks using your familiar iOS vocabulary so that you immediately understand, without lengthy explanations, because you already know the mobile subjects from being an iOS developer.

You don't need experience in the Java language, although it does help a lot. The most important qualities of Android developers do not include Java programming language experience. It is the mobile SDK and framework knowledge that distinguishes you from other Java programmers. You know one programming language already since you are an iOS developer, so you should be comfortable reading Java code. I also made the sample code extra-readable, so you will have no problem following through the programming subjects and the Java sample code.

There are tons of Java language references out there; you should find them handy sooner or later when you are ready to get serious.

How This Book Is Organized

In Part I, you will get the Android development toolkit up and running in no time. With the Android IDE, 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 Android apps by following the same iOS thinking process: create a storyboard and break the app into model-view-controller (MVC) classes. You will be able to reuse most of the existing software artifacts and design from the iOS counterparts. The common mobile topics are followed, including user interface, managing data, and networking with remote services. After you finish Part II, you will be able to create simple but meaningful Android apps with rich UI components, and to handle common CRUD (create, read, update, delete) operations locally and remotely. There are still more Android goodies to come.

In Part III, this book recaps the Android framework fundamentals with code instead of just descriptions. You will discover the uniqueness of the Android framework and appreciate many features that you normally don't have in iOS. Several powerful and repeatable mobile UX patterns are also introduced. Once you get here, you should be fully convinced that you can do everything in Android just like you do in iOS. The last chapter walks you through a case study that ports a complete iOS app to Android. It recaps how to use the iOS analogies and mapping guidelines from the topics in previous chapters. You can also use the book's table of contents to help find the porting guidelines as needed.

When you complete the journey, you will be able to use the right tools to effectively port your existing iOS apps to Android.

Part 1

Prepare Your Tools

A handy tool makes a handy man. This is very true for creating software, too. You use Xcode to write, compile, debug, and build code; it is an Integrated Development Environment (IDE) for iOS programming. ADT in Eclipse is an IDE for Android programming, which offers comparable tools and features as Xcode. The first part of the chapter walks you through the installation options and steps for getting it up and running. All the topics in this book come with sample code. You will need to use the IDE to learn from these sample projects and you will use the IDE to create world-class Android apps, too.

For native Android programming, Java is the designated programming language. The chapters in this part will give you enough knowledge to read the sample code, in case you are not exposed to the Java language yet. You will feel comfortable using the code from this book as your own code, without assuming you already know Java.

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 that we can use it—and learn Android programming along the way.

The Android Developer Tools Plugin for Eclipse

IOS ANALOGY

Just as Xcode is an IDE for creating iOS apps, the ADT plugin for Eclipse is an IDE for creating Android apps.

The *Android Developer Tools (ADT) plugin for Eclipse* is the Android-programming integrated development environment (IDE) that we will go over in detail. It is a full Java IDE that includes the Android SDK to help you build, test, debug, and package your Android apps. It is free, open source, and runs on most major operating system (OS) platforms, including the Mac OS. The ADT plugin is not a developer's only choice, but probably the one most commonly used. We will use it throughout this book.

Installing the All-in-One Bundled Package

The all-in-one bundled package is the best option for most Android developers. It is similar to Xcode installation: there is no need to sort out the dependencies and no need for manual configurations. It actually wasn't available when I started Android programming a couple years ago.

With a single download, the ADT Bundle includes everything you need, including the following:

- Eclipse
- The ADT plugin for Eclipse
- Android SDK Tools
- Android Platform-tools
- The latest Android platform
- The device emulator image for the latest Android platform

Get the single download for your Mac at http://developer.android.com/sdk/index.html#mac-bundle.

It is a ZIP file. Unzip it and put the contents of the ZIP file anywhere you want; for example, if you put it in `/Applications/adt-bundle-mac`, it should look like Figure 1-1.

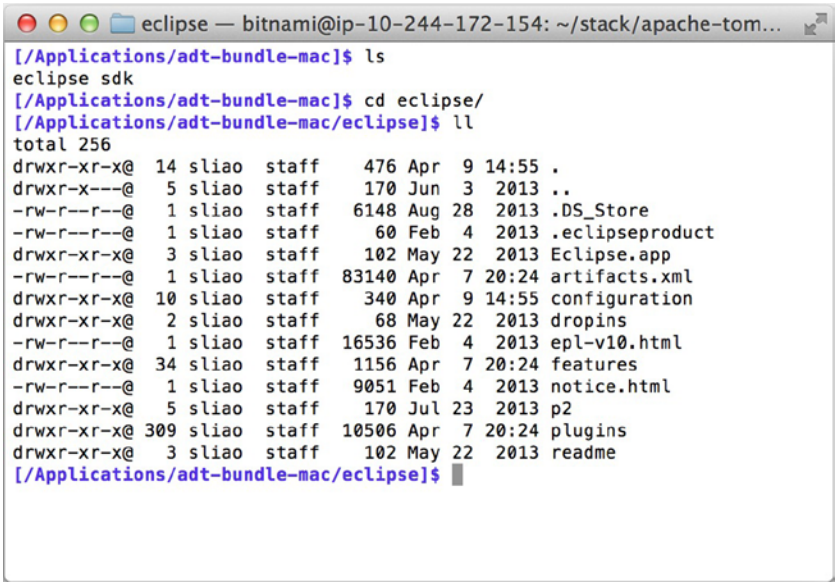

Figure 1-1. *The adt-bundle-mac folder structure*

All you need to do is download it and unzip it. Please allow me to repeat: you won't need to configure it after you install it. Go ahead and launch the Eclipse.app. Let's keep it in the Mac OS Dock so that you can launch it at any time.

If you don't want to mess with the ADT plugin, you may choose to skip the next section and go straight to the "MacBook Retina Display" section.

Installing the Eclipse ADT Plugin

You may choose to manually install the components in the ADT Bundle and go through the configuration steps. The ADT plugin for Eclipse is a custom plugin for the Eclipse IDE that provides an integrated environment to develop Android apps. It extends the capabilities of Eclipse to let you quickly set up new Android projects, build an app user interface (UI), debug your app, and create app packages (APKs) for distribution.

I installed and configured the ADT plugin manually because I also use Eclipse for JavaEE programming, and I want to share common Java classes between JavaEE server code and Android client code. If you choose to install the plugin to your existing Eclipse instance, chances are you already have experience with the Eclipse IDE.

If you are not a JavaEE developer or just want to keep things simple for now, you should skip the following instructions and do the all-in-one bundled package installation.

> **Note** Even if you already have Eclipse installed, you still can have multiple Eclipse instances. You should only need to go through this plugin option if you need to share Java classes between Android projects and J2EE projects.

If you decide to go with the manual ADT plugin option for your existing Eclipse app, please visit the Android official site (http://developer.android.com/sdk/installing/installing-adt.html) for detailed instruction.

For the Mac OS, you can also follow the "Installing the ADT Plugin Cheat Sheet," which is modified from the preceding URL for your convenience.

INSTALLING THE ADT PLUGIN CHEAT SHEET

Do the following to download the ADT plugin:

1. Open the Install wizard from the Eclipse top menu bar by selecting **Help ➤ Install New Software** (see Figure 1-2). Click **Add** to add a new site.

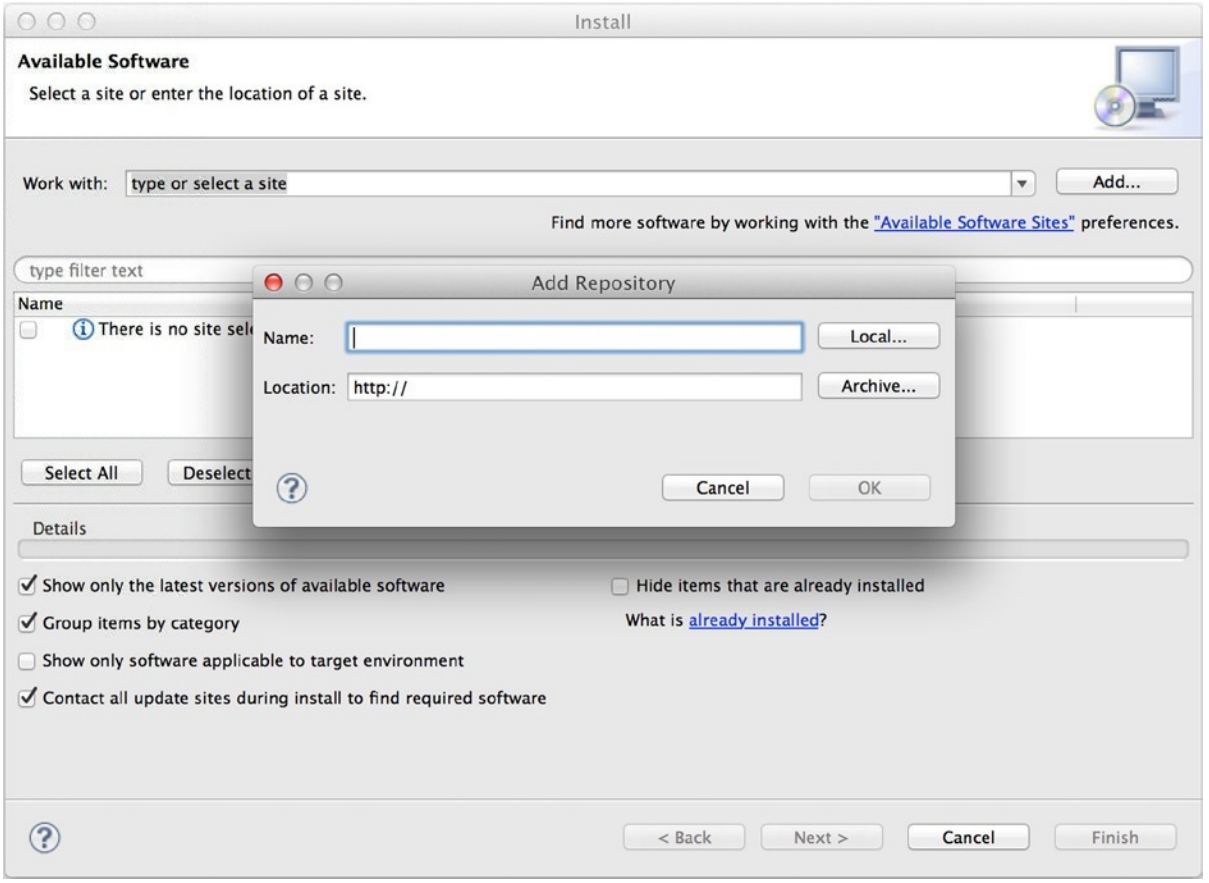

Figure 1-2. Install New Software ➤ Add Repository

2. In the **Add Repository** dialog, enter the following and then click **OK**.
   a. Enter **ADT Plugin** in the **Name** field.
   b. Enter **https://dl-ssl.google.com/android/eclipse/** in the **Location URL**.

> **Note** If you have trouble acquiring the plugin, try using "http" in the Location URL instead of "https".

3. In the **Work with** drop-down, make sure the ADT Plugin repository is selected (see Figure 1-3).

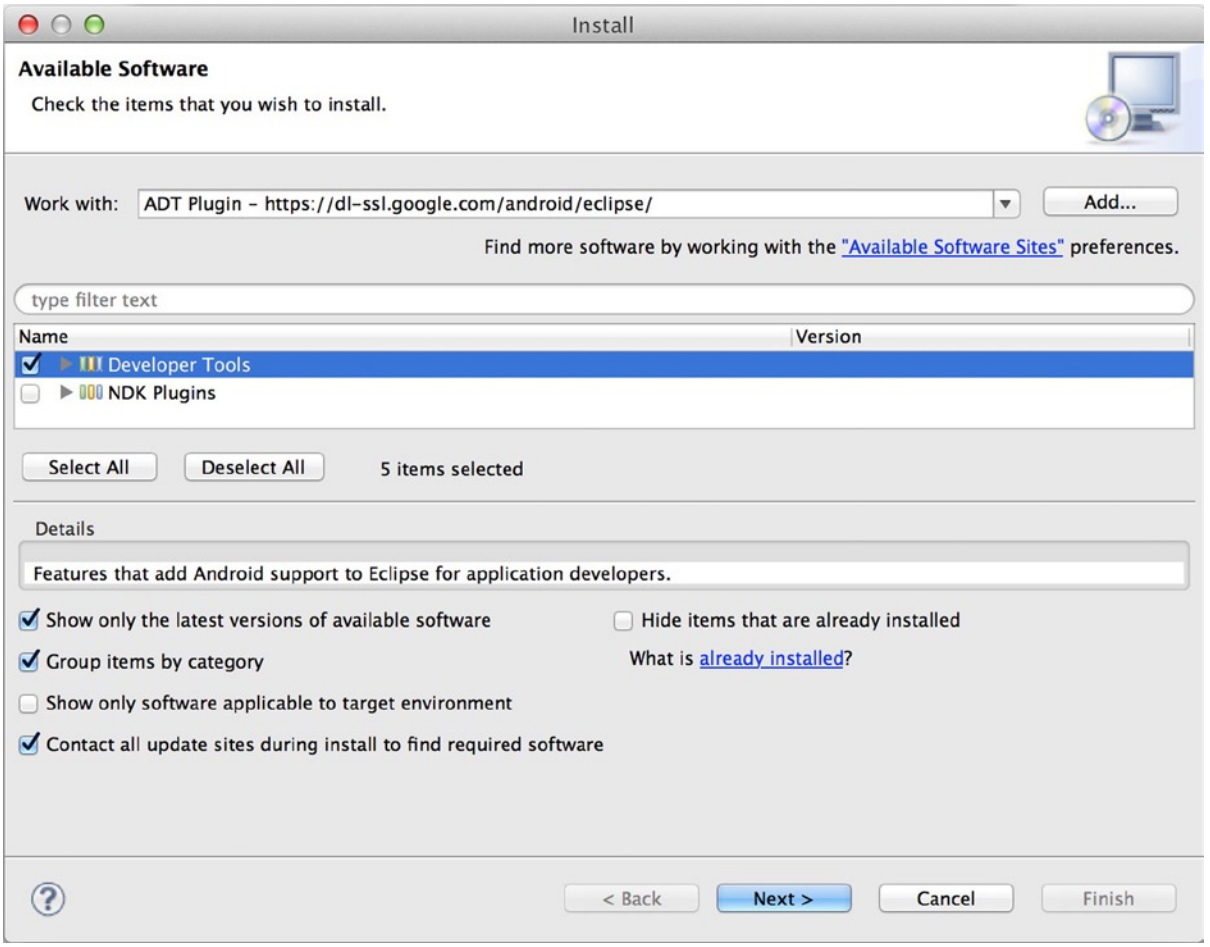

Figure 1-3. *Select Developer Tools*

a. Select the **Developer Tools** check box and then click **Next**.

4. You should see a list of the tools to be downloaded. Click **Next** and follow the onscreen instructions to complete the installation. When the installation completes, restart Eclipse.

Do the following to specify the Android SDK location:

1. Get the Android SDK, which contains the framework libraries and the development toolkit.

   a. Download the ZIP file from `http://developer.android.com/sdk/index.html#download`.

   b. Unzip the file to your **SDK Location** folder, which is /Applications/adt-bundle-mac/sdk.

2. From the Eclipse top menu bar, select **Eclipse ➤ Preferences…** to open the Eclipse Preferences screen (see Figure 1-4). Select **Android** from the left panel and enter your SDK Location: **/Applications/adt-bundle-mac/sdk**.

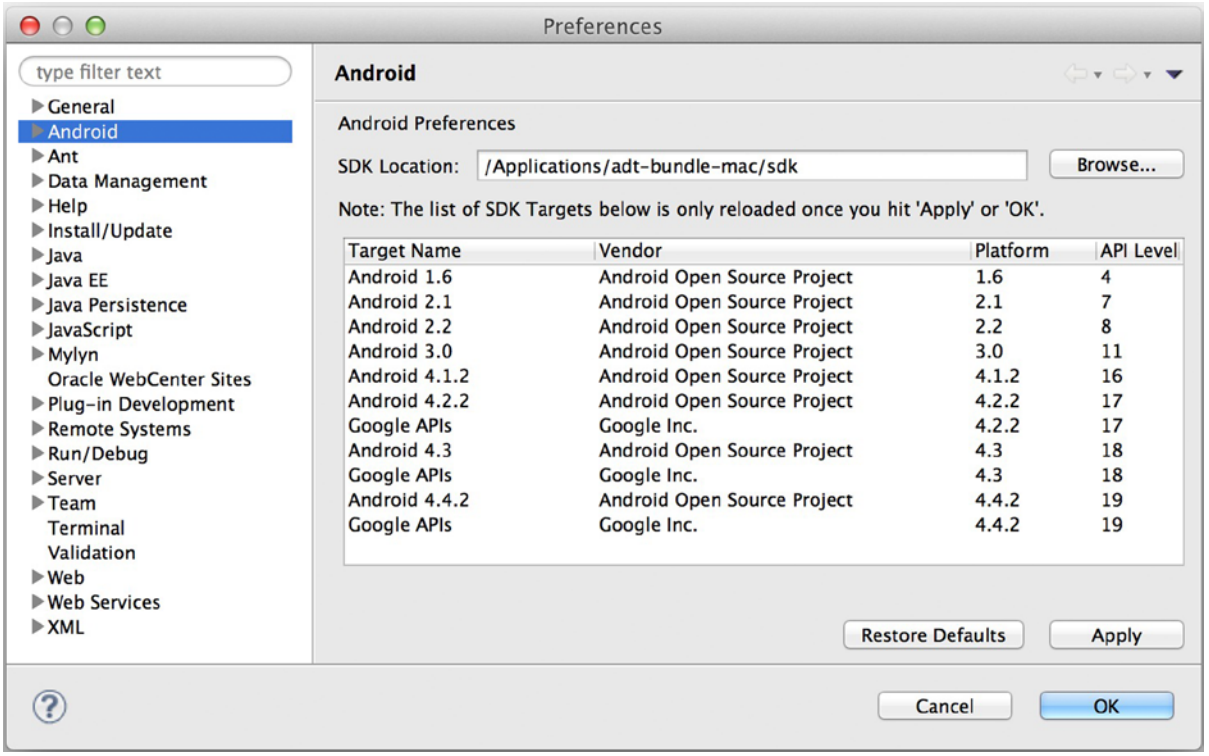

Figure 1-4. *Android SDK Location*

Do the following to add platforms and packages:

1. You need at least one platform-specific SDK and tools. Use the Android SDK Manager to obtain them for the latest platform (API 19 is the latest as of this writing).
2. From the Eclipse top menu bar, select **Window ➤ Android SDK Manager** to launch the Android SDK Manager. Figure 1-5 shows the platform API and tool packages that I have installed.

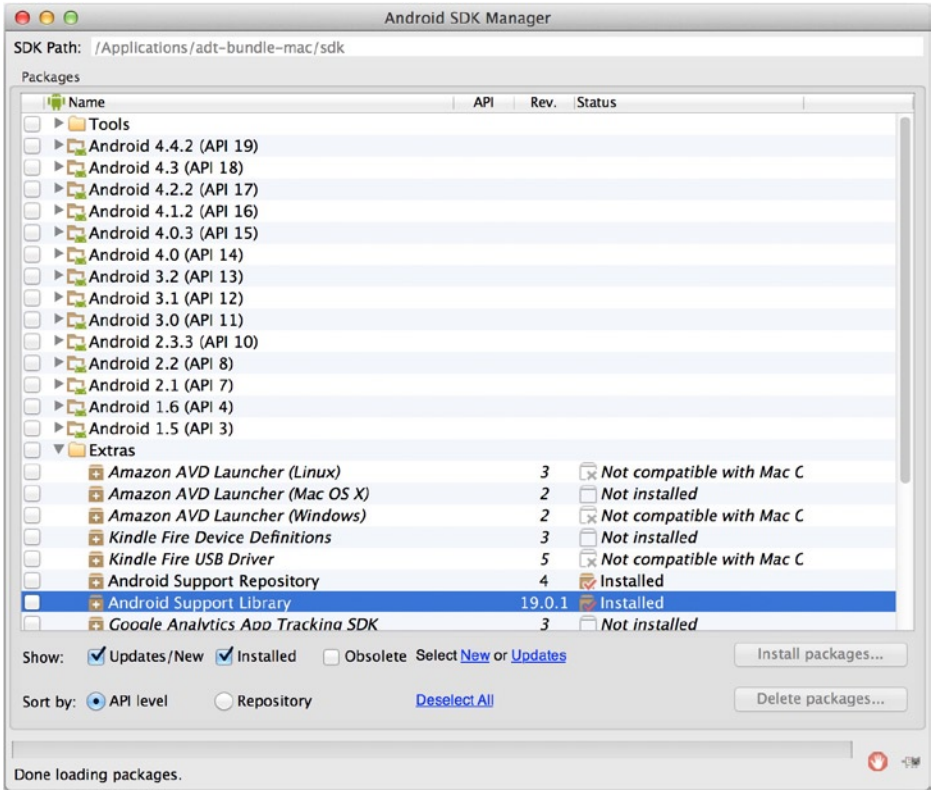

***Figure 1-5.** Android SDK Manager*

3. You want to install at least the following three packages (actually the first two, but you will need the last one later):

    a. Tools

    b. The latest Android API: Android 4.4.2 (API 19) as of this writing

    c. Extra: The Android Support Library

Follow the onscreen instructions to complete the installation.

MacBook Retina Display

If you have a Mac with a beautiful retina display, the Eclipse screen resolution looks awfully bad because it is not yet optimized for retina display. My eyes simply cannot stand the low resolution on my retina screen. It really has zero effect on any code you write, but please do the following so that you don't waste the beauty of a retina display:

1. In the Finder, right-click **Eclipse.app** and then select **Show Package Contents**.
2. Modify the Info.plist file with any text editor, such as TextEdit.app. If you have Xcode, double-clicking will open the file in the Xcode editor as well. Note that your modifications (see Listing 1-1) go inside `<dict> </dict>`.

   ***Listing 1-1.** Eclipse Info.plist File*

   ```
   <plist version="1.0">
   <dict>

         ...
      <key>NSHighResolutionCapable</key>
         <true/>
         ...

   </dict>
   </plist>
   ```

3. Force your Mac OS to reload the preceding changes the next time you launch Eclipse.app. This can be easily achieved using the shell `touch` command.
4. Open the Terminal program and issue the `touch` command, as shown in Listing 1-2.

   ***Listing 1-2.** The Touch Command*

   ```
   [/Applications/adt-bundle-mac/eclipse]$ cd /Applications/adt-bundle-mac/eclipse
   [/Applications/adt-bundle-mac/eclipse]$ ls
   Eclipse.app        configuration        epl-v10.html        notice.html       plugins
   artifacts.xml      dropins              features            p2                readme
   [/Applications/adt-bundle-mac/eclipse]$ touch Eclipse.app/
   ```

Bingo! Eclipse should have a retina display now.

The Eclipse Workbench

You just got the right tool, but you need to know how to use it. Let's spend some time with Eclipse first because it appears quite different from Xcode. I think it is actually more sophisticated than Xcode because it has a broader goal: it provides a plugin platform so that you can extend the IDE by creating a plugin for your unique development tasks. For example, the ADT plugin for Android that you just installed is a plugin toolkit for Android development. In the Java world, you can use Eclipse

for almost any Java solutions, including JavaEE, JavaME, the Blackberry SDK, various third-party vendor solutions, or SDKs. There is also a C/C++ plugin called Eclipse CDT for the C/C++ toolchain and make utility. There is an Eclipse plugin for the Symbian mobile development toolkit as well.

The Eclipse Workspace

Enough of the motivational talk, let's start using Eclipse and create a workspace.

IOS ANALOGY

Same as the Xcode workspace idea, the Eclipse workspace is a logical grouping of related projects; however, the Eclipse workspace needs to be a physical folder.

Please complete the following steps to create an Eclipse workspace:

1. Launch Eclipse and enter a folder name for the **Workspace** (see Figure 1-6), such as `/Users/sliao/Documents/adtWs`.

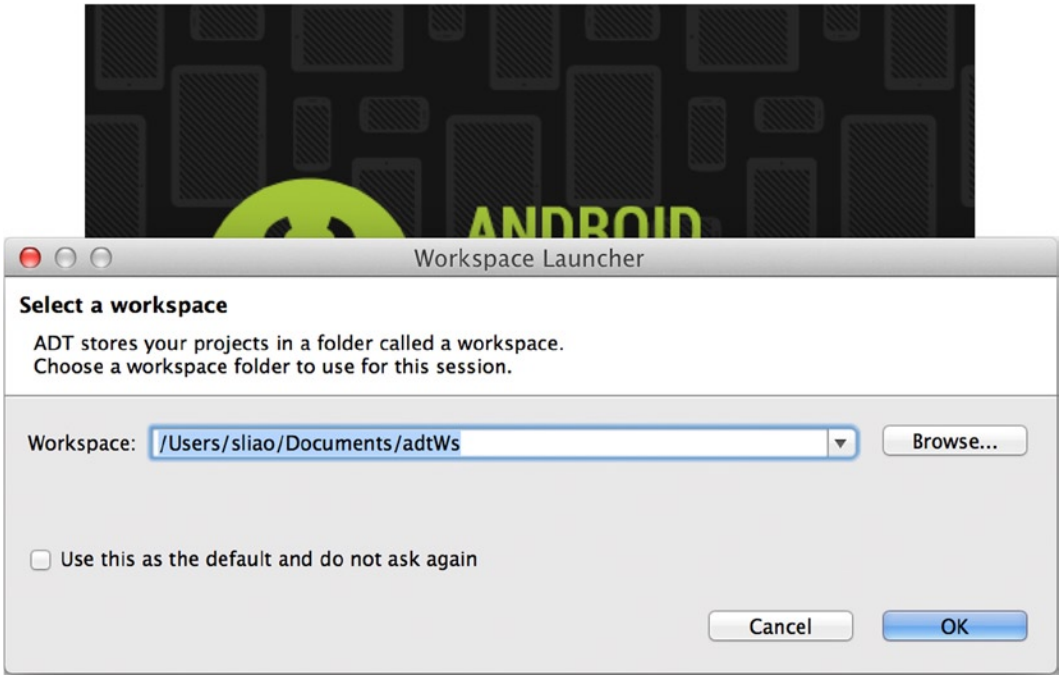

Figure 1-6. *The Eclipse Workspace Launcher*

2. Click **OK**. You will get the Welcome! screen (see Figure 1-7) the first time you create a new workspace. Let's close it for now. You can always get back to this screen from the top menu bar by selecting **Help ➤ Android IDE**.

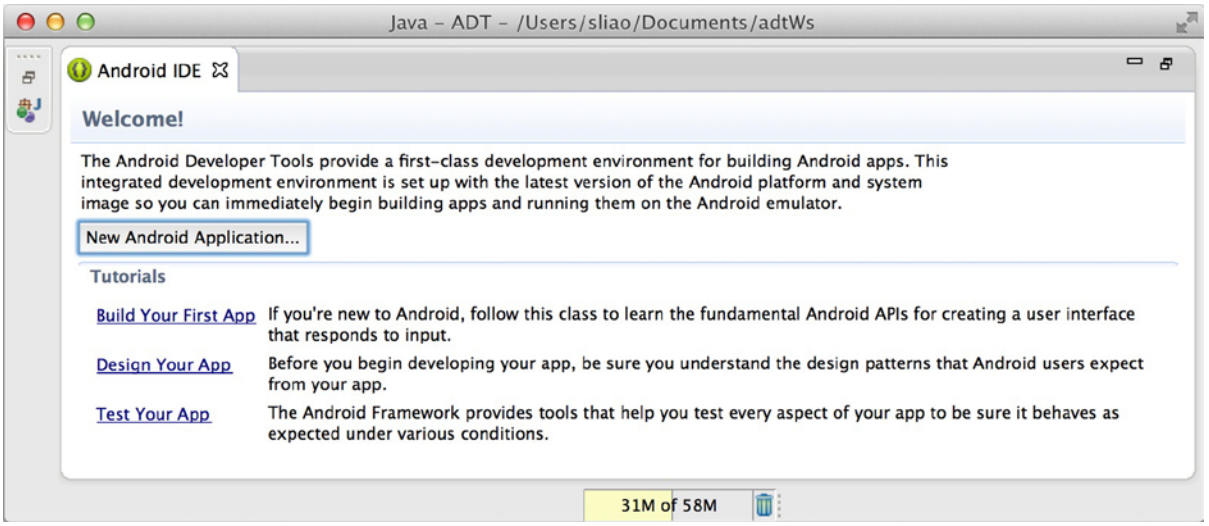

Figure 1-7. *The ADT Welcome! screen*

A single Workbench window is displayed, as shown in Figure 1-8. A Workbench window offers one or more perspectives. A *perspective* contains editors and views, such as the Package Explorer. Upon the Workbench window launching for the first time, the Java perspective is displayed. Good, we need a Java perspective because Java is the language for Android programming. You can always click the Java button in the perspectives toolbar to switch to the Java perspective (see Figure 1-8).

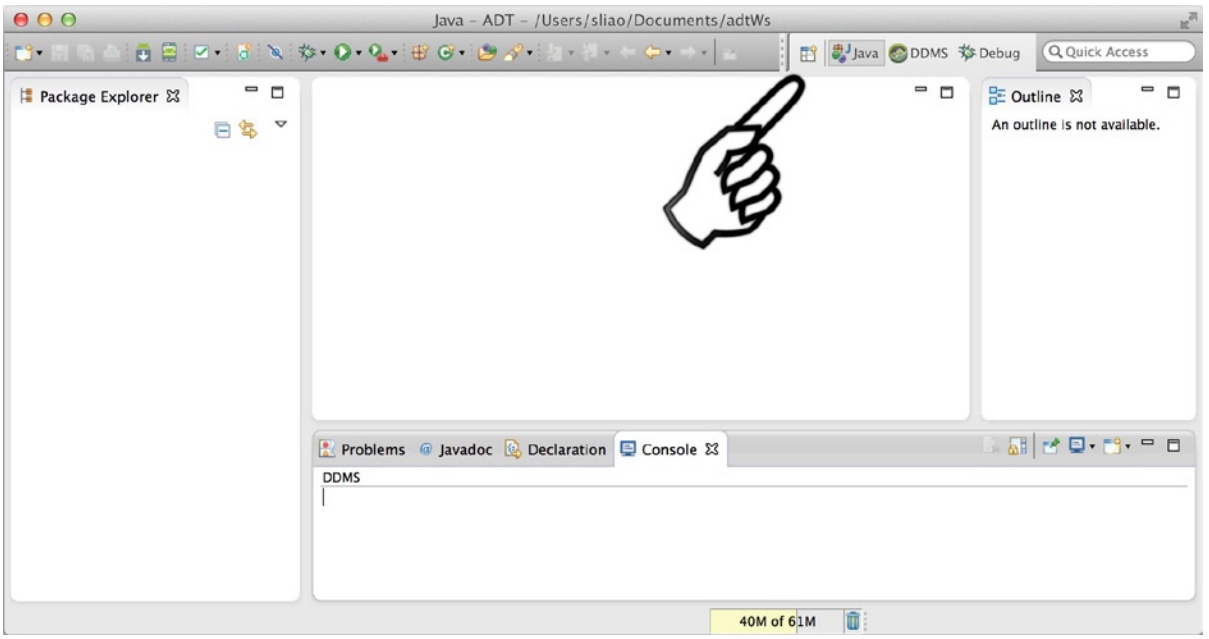

Figure 1-8. *The Eclipse Workbench*

3. Go ahead and play with this window to satisfy your curiosity if you wish. Don't be afraid of messing up anything. For example, there are many options in the menu bar and a few buttons on the toolbar. The context menu always contains hidden gems. You can get back any time by going to the menu bar and selecting **Windows ➤ Reset Perspective...**.

Learning Eclipse has been one of my best investments of time. It is the only tool that I am still using since Y2K. I did not get many chances to reuse the same tool for other technologies. If you have been in IT for a long time, you know what I am talking about.

Eclipse can do a lot, more than the Android IDE. But you don't need to know everything today. After all, it is a tool, and you naturally get better at it when you use it often. We will focus on using it for Android programming.

Create an Android Project Using the Template

You just got the right tool, and it is up and running. Wouldn't you like to see some real action—like creating an Android app and see it running? I'd like that, too! You can make sure your IDE is working properly as well.

My very first Xcode app was actually created using the **Create a new Xcode project** template (see Figure 1-9) when I had no idea what Objective-C looked like. All I wanted was to see something running in no time. Yep, Xcode did it for me nicely. I was very happy with myself when I felt I created an iOS app without knowing anything! Hey, there is nothing wrong with making yourself happy right?

The Eclipse with ADT plugin offers the same thing.

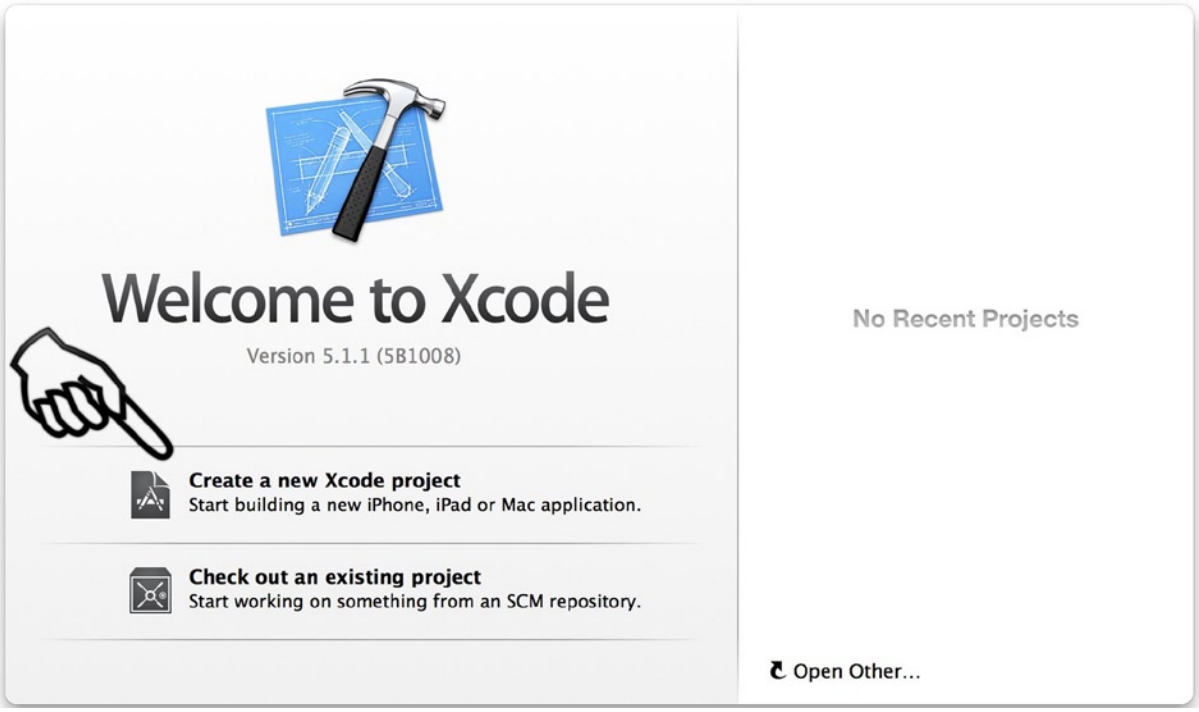

Figure 1-9. Create a new Xcode project

The objective of this lesson is to create an Android app as quickly as possible. Let's hold any programming questions for now to finish the project as fast as you can. Please complete the following steps:

1. Launch the **Eclipse.app** if it is not launched yet.
2. Open the **Android Application wizard** from the Eclipse top menu bar and select **File ➤ New ➤ Android Application Project** (see Figure 1-10).

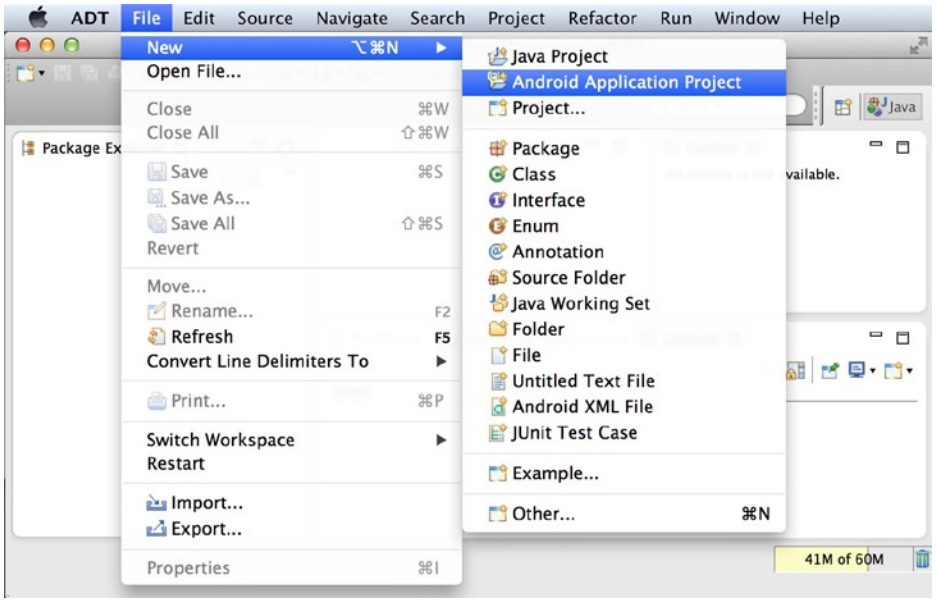

Figure 1-10. *Create a new Android Application project*

3. Do the following in the **New Android Application section** (see Figure 1-11), and then click **Next**.
   a. Enter **LessonOne** in the **Application Name** field.
   b. Select the latest SDK, which is the last option on the **Minimum Required SDK** drop-down menu. The latest SDK package is preinstalled with the bundled ADT; otherwise, you might need to install the corresponding SDK version.
   c. Accept the default values for the remaining fields.

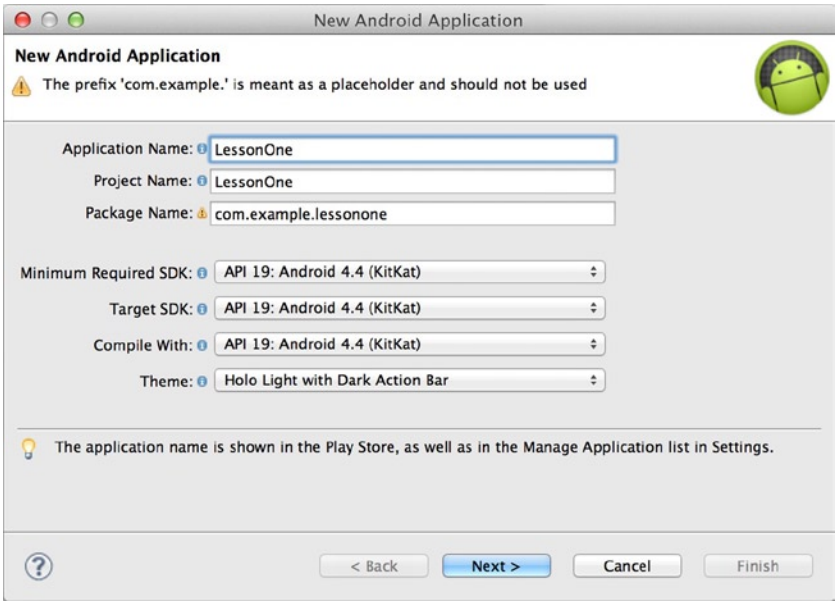

Figure 1-11. New Android Application section

4. On the next screen, keep all the default values in the Configure Project section (see Figure 1-12). Click **Next**.

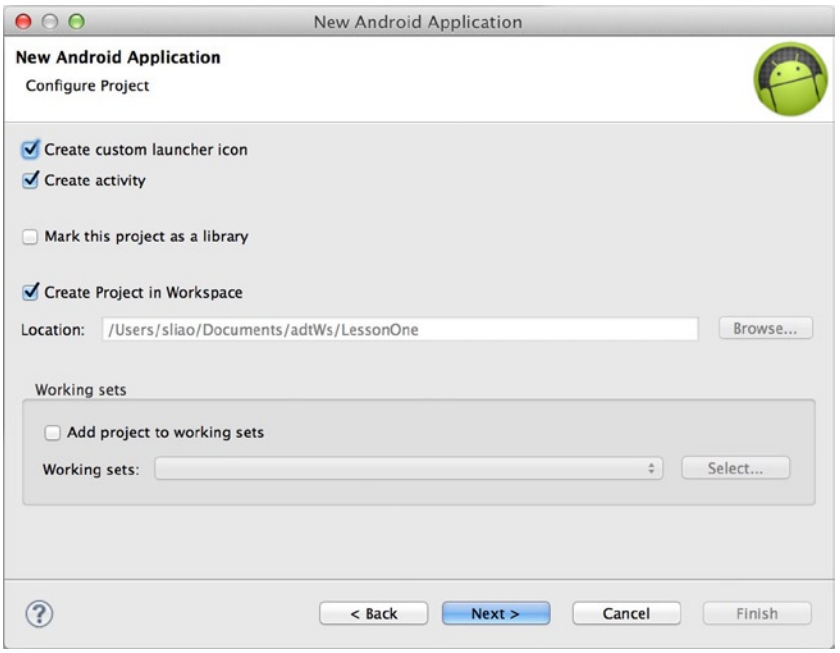

Figure 1-12. Configure Project

5. You can supply an optional launcher icon on the next screen. Click **Next** to use the default Android robot icon (see Figure 1-13).

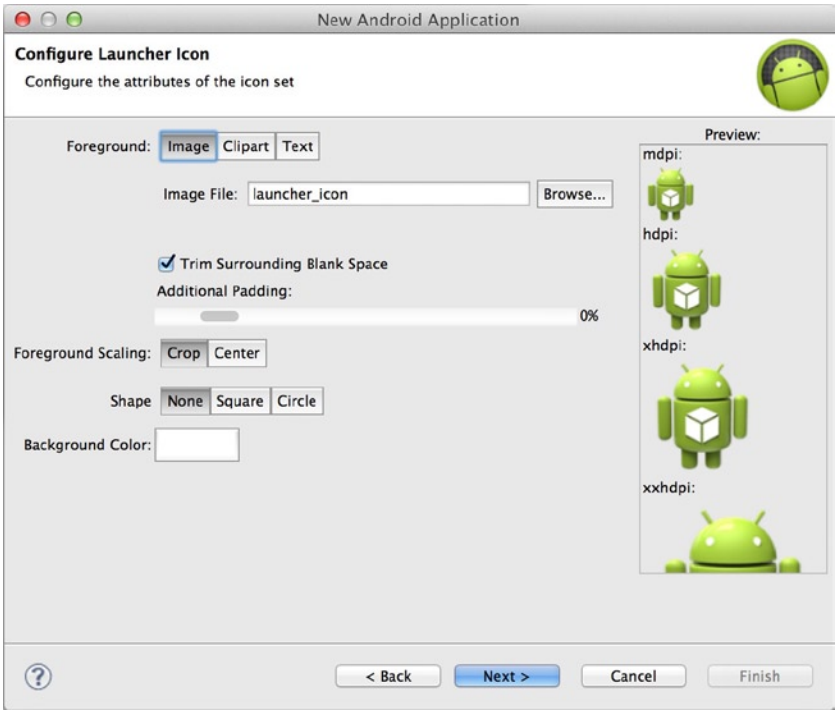

Figure 1-13. *Configure Launcher Icon*

6. The next screen is Create Activity (see Figure 1-14), where you do the following:

   a. Select the **Create Activity** check box.

   b. Select **Master/Detail Flow**. It is the most sophisticated template among the three choices, making it more fun to play with this app later. Click **Next** when done.

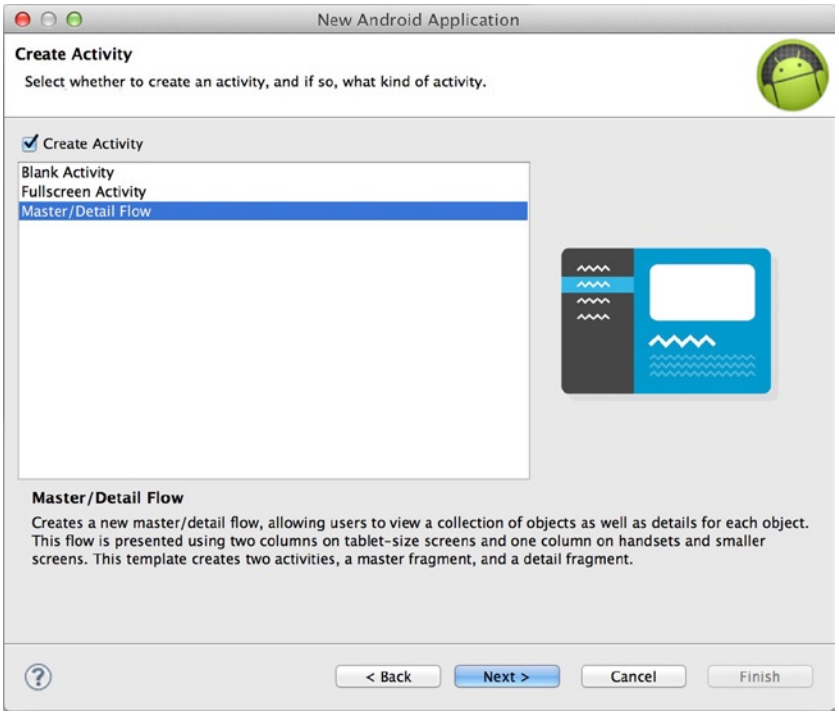

Figure 1-14. Create Activity with templates

7. Keep the prefilled values on the following screen (see Figure 1-15). They will not be used because you are not going to write any Java code yet. Click **Finish**.

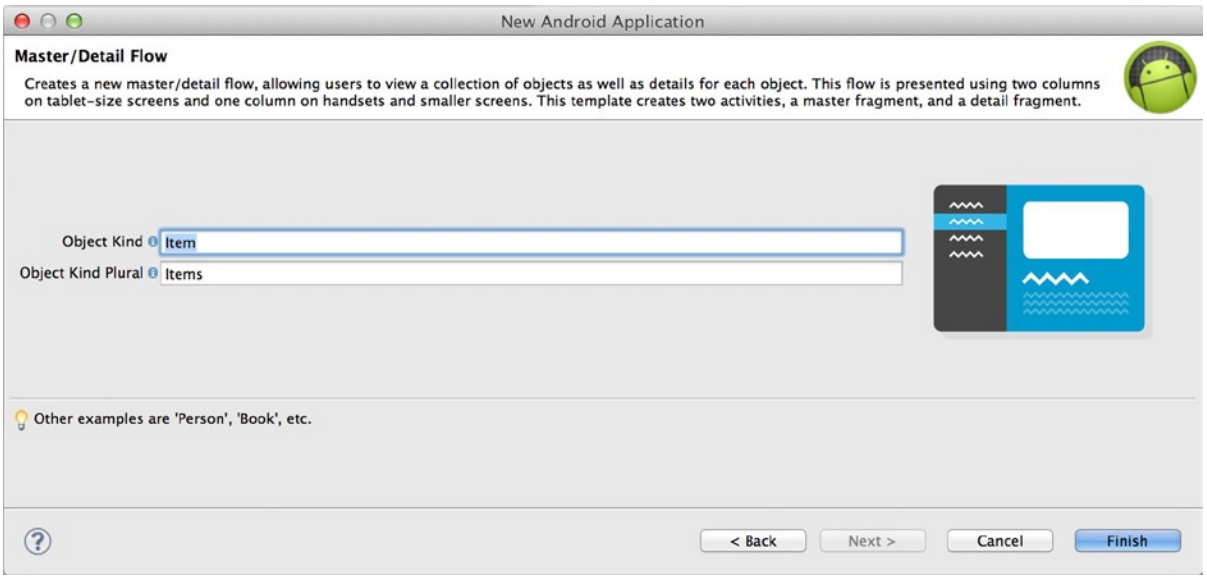

Figure 1-15. Java class for list items

You just created an ADT project, the LessonOne project. It should appear in the Package Explorer view (see Figure 1-16). Just like using Xcode project creation templates, the ADT New Android Application wizard creates the ADT project folder, the application source code, and all the resources for building the template apps.

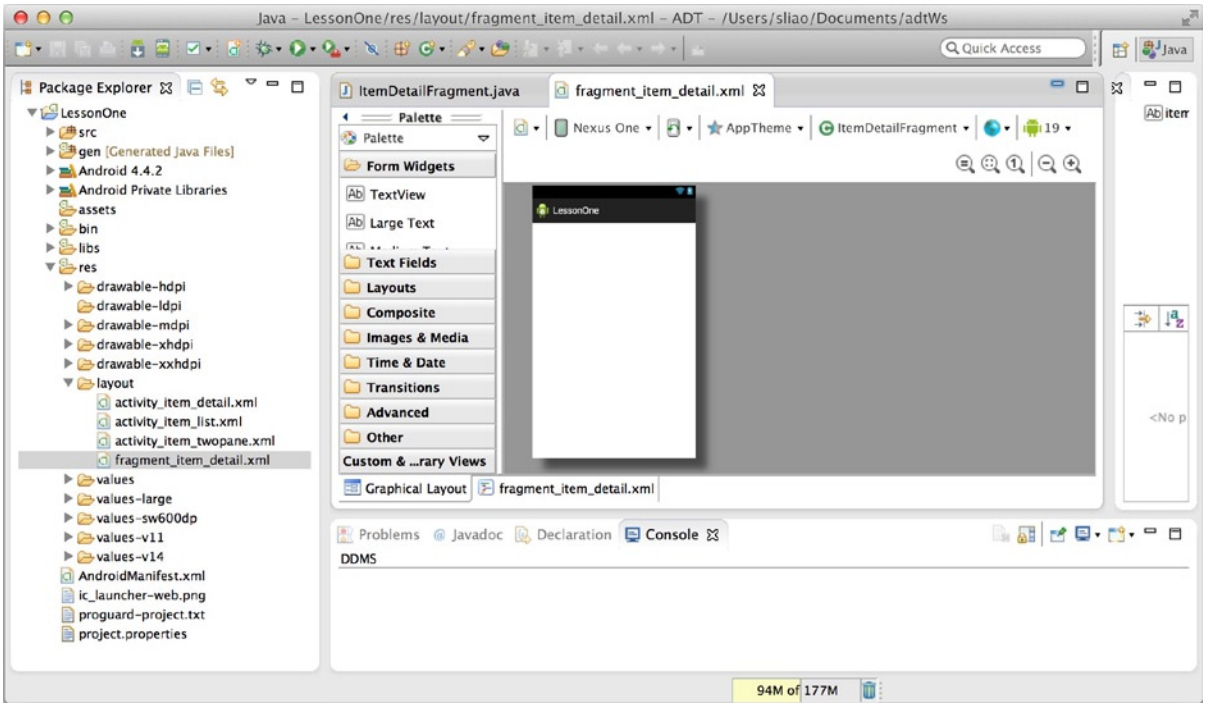

Figure 1-16. *LessonOne project in Project Explorer*

As a bonus, create two more projects using the other two templates (see Figure 1-14). Figure 1-17 shows three projects in the Package Explorer view.

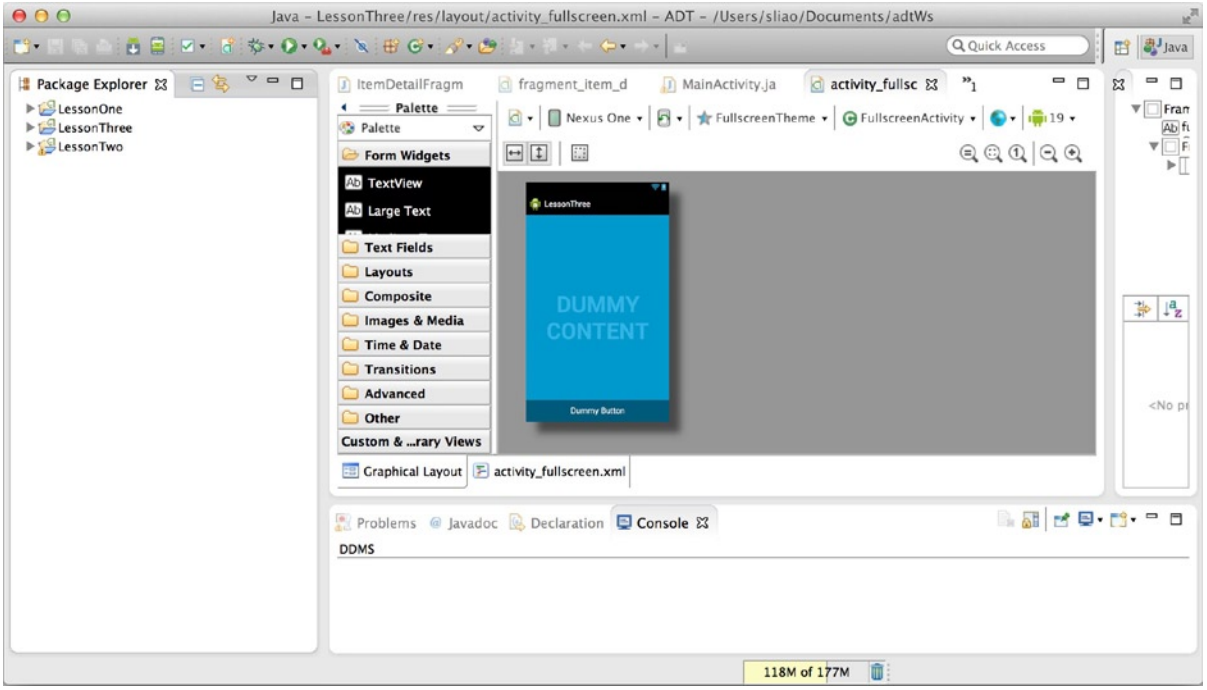

Figure 1-17. Three projects in Project Explorer

Build the Project

IOS ANALOGY

The Xcode Build action keyboard shortcut on the Mac is Command-B (⌘B).

The Eclipse workbench is set to *Build Automatically* by default (see Figure 1-18); you don't need to build ADT projects explicitly. Let's keep this option; I just wanted to point out that you can disable this option and do a manual build (⌘B) if you wish to.

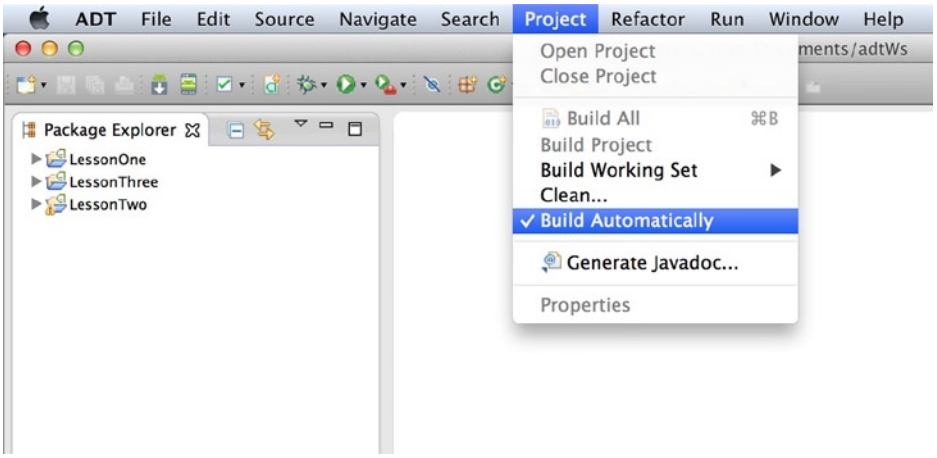

Figure 1-18. *Build Automatically*

When I have a lot projects in the workspaces, I may choose to disable the automated build option to avoid the build actions that kick in automatically. This takes up a lot of your CPU whenever any resource is being saved. When this feature annoys me, I turn it off.

Launch the App

IOS ANALOGY
The Xcode Run action keyboard shortcut on the Mac is Command-R (⌘R).

The LessonOne project should have no errors. You can launch the app and see it run on device emulators or on Android-powered devices. The emulator is a very important piece of any IDE. In the following steps, you will prepare an Android Virtual Device (AVD) and launch the LessonOne project onto a device emulator.

1. From the LessonOne project context menu, select **Debug As ➤ Android Application** (see Figure 1-19).

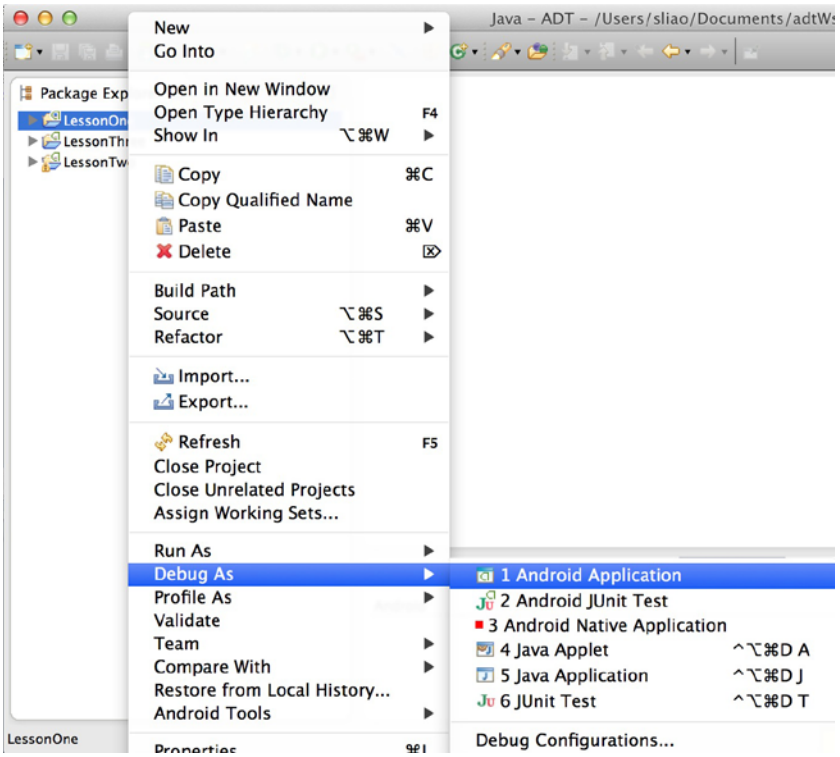

Figure 1-19. Debug As Android Application

2. You most likely will get to the following screen, Android AVD Error (see Figure 1-20), since you haven't created any Android Virtual Device (AVD) yet. You can click the **Yes** button to get to the Android Virtual Device Manager tool. However, I want to show you another path so that you can create an AVD any time you want to. Click **No** to close the error dialog.

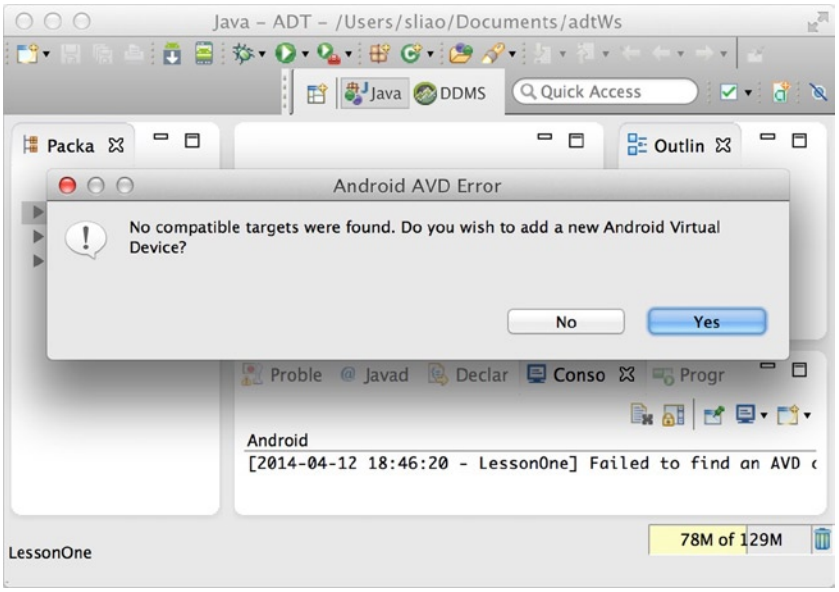

Figure 1-20. Android AVD Error: No compatible targets were found

3. In the ADT top menu bar, select **Window ➤ Android Virtual Device Manager**. Figure 1-21 shows my AVD Manager without any device.

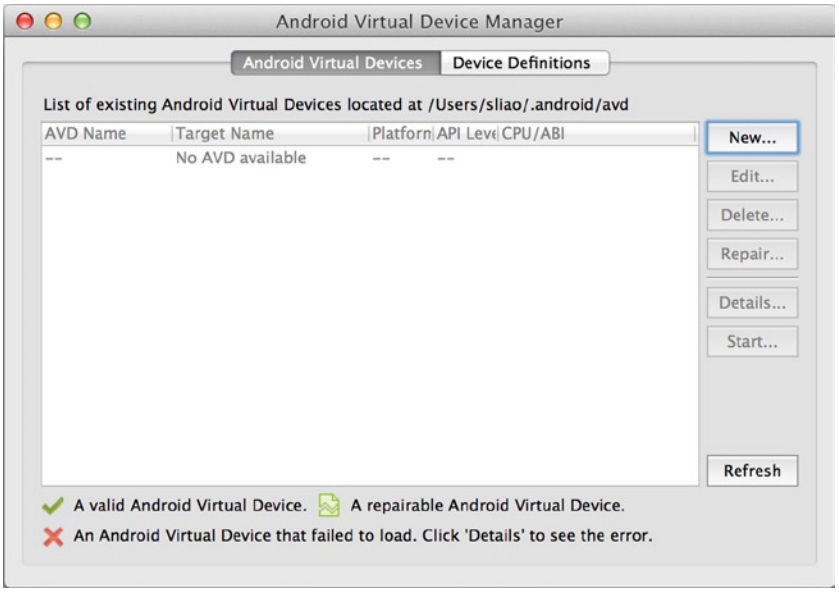

Figure 1-21. AVD Manager

Note You can select New, Edit, or Delete for any existing AVD. Repair does not seem to work for me. I simply delete and create a new one instead. The Device Definitions button shows the preset AVD definitions; you will find this useful when you want to test your app and know the specs.

4. Click the **New. . .** button to continue creating your first AVD (see Figure 1-22). Leave everything as the default, except the following:
   a. Enter any name, such as **nexus7**, in the **AVD Name** field.
   b. Select **Nexus 7** from the **Device** drop-down menu.
   c. Select the highest API level from the **Target** drop-down menu. This is the most significant attribute. The Target needs to be compatible with the Minimum Required SDK specified for your app in Figure 1-11, which should be what you entered when creating the LessonOne project.
   d. Optionally, enter information for any other fields with your app's required specific hardware. For example, if your app saves data on external storage, you will experience an error if information in the SD Card section is not specified.

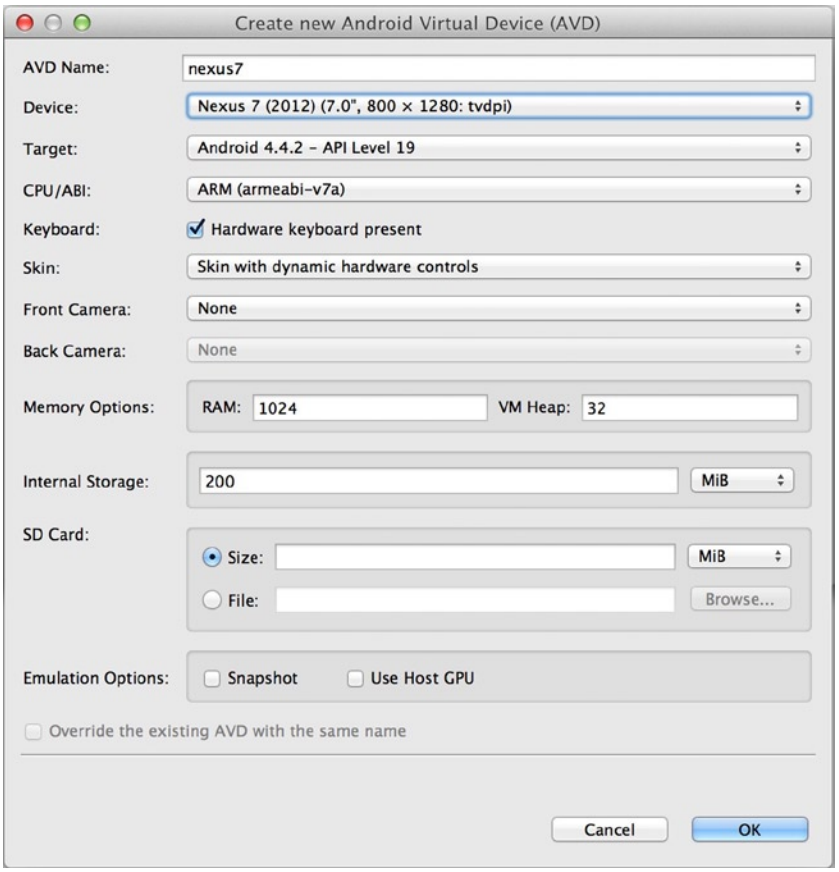

Figure 1-22. Create AVD

5. Figure 1-23 shows the newly created AVD, which is compatible with the LessonOne project.

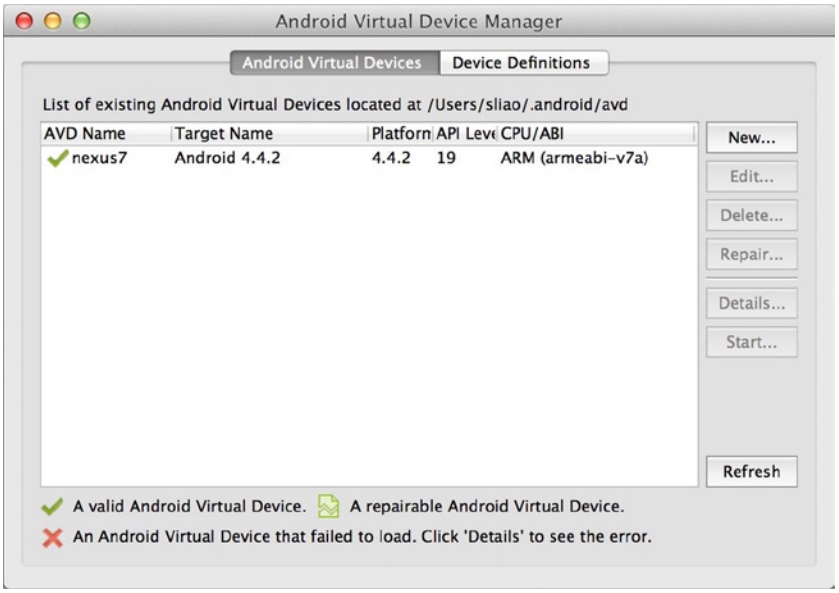

Figure 1-23. *AVD list*

6. Dismiss the AVD Manager and relaunch the LessonOne project. You should see the app running in the newly created emulator (see Figure 1-24).

Figure 1-24. *LessonOne app in emulator*

> **Note** I always experience a timeout error on the first try. You might experience the same error if your machine is not fast enough. The newly created AVD seems to take too long to start up. If you get an error message, relaunch again *after* the emulator has started.

Play with the app and the emulator. A mouse-click event on an emulator is equivalent to a touch event on a real Android device. If you don't have a device yet, definitely play with the emulator to get familiar with the emulated Android device.

> **Tip** Rotate the emulator (fn+Ctrl+F12) in landscape mode to see how the LessonOne app does. Check `http://developer.android.com/tools/help/emulator.html#KeyMapping` for the emulator keyboard mappings.