Lauren Darcey
Shane Conder

Third Edition
Features Android 4.0

Android™
Wireless Application Development
Volume I: Android Essentials

Developer's Library

Android™ Wireless Application Development

Volume 1: Android Essentials

Third Edition

This page intentionally left blank

Android™ Wireless Application Development

Volume 1: Android Essentials

Third Edition

Lauren Darcey
Shane Conder

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data:
Darcey, Lauren, 1977-
Android wireless application development / Lauren Darcey, Shane Conder.
v. cm.
Includes bibliographical references and index.
Contents: v. 1. Android essentials
ISBN 978-0-321-81383-1 (pbk. : alk. paper)
1. Application software–Development. 2. Android (Electronic resource) 3. Mobile computing. I. Conder, Shane, 1975- II. Title.
QA76.76.A65D258 2012b
005.1–dc23

2011049390

ISBN-13: 978-0-321-81383-1
ISBN-10: 0-321-81383-9

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

First Printing: February 2012

Editor-in-Chief
Mark Taub

Acquisitions Editor
Trina MacDonald

Development Editor
Songlin Qiu

Managing Editor
Kristy Hart

Project Editor
Lori Lyons

Copy Editor
Bart Reed

Indexer
Heather McNeill

Proofreader
Paula Lowell

Technical Reviewers
Douglas Jones
Mike Wallace
Mark Gjoel

Publishing Coordinator
Olivia Basegio

Multimedia Developer
Dan Scherf

Book Designer
Gary Adair

Compositor
Nonie Ratcliff

❖

This book is dedicated to Chickpea.

❖

Contents at a Glance

Table of Contents

II: Android Application Basics

III: Android User Interface Design Essentials

IV: Android Application Design Essentials

Acknowledgments

This book would never have been written without the guidance and encouragement we received from a number of supportive individuals, including our editorial team, coworkers, friends, and family. We'd like to thank the Android developer community, Google, and the Open Handset Alliance for their vision and expertise. Throughout this project, our editorial team at Pearson Education (Addison-Wesley) always had the right mix of professionalism and encouragement. Thanks especially to Trina MacDonald, Olivia Basegio, Songlin Qiu, and our crack team of technical reviewers: Doug Jones, Mike Wallace, and Mark Gjoel, (as well as Dan Galpin, Tony Hillerson, Ronan Schwarz, and Charles Stearns, who reviewed previous editions). Dan Galpin also graciously provided the clever Android graphics used for Tips, Notes, and Warnings. We'd also like to thank Ray Rischpater for his longtime encouragement and advice on technical writing. Amy Badger must be commended for her wonderful waterfall illustration, and we also thank Hans Bodlaender for letting us use the nifty chess font he developed as a hobby project.

About the Authors

Lauren Darcey is responsible for the technical leadership and direction of a small software company specializing in mobile technologies, including Android, iOS, Blackberry, Palm Pre, BREW, and J2ME and consulting services. With more than two decades of experience in professional software production, Lauren is a recognized authority in application architecture and the development of commercial-grade mobile applications. Lauren received a B.S. in Computer Science from the University of California, Santa Cruz.

She spends her copious free time traveling the world with her geeky mobile-minded husband and is an avid nature photographer. Her work has been published in books and newspapers around the world. In South Africa, she dove with 4-meter-long great white sharks and got stuck between a herd of rampaging hippopotami and an irritated bull elephant. She's been attacked by monkeys in Japan, gotten stuck in a ravine with two hungry lions in Kenya, gotten thirsty in Egypt, narrowly avoided a coup d'état in Thailand, geocached her way through the Swiss Alps, drank her way through the beer halls of Germany, slept in the crumbling castles of Europe, and gotten her tongue stuck to an iceberg in Iceland (while being watched by a herd of suspicious wild reindeer).

Shane Conder has extensive development experience and has focused his attention on mobile and embedded development for the past decade. He has designed and developed many commercial applications for Android, iOS, BREW, Blackberry, J2ME, Palm, and Windows Mobile—some of which have been installed on millions of phones worldwide. Shane has written extensively about the mobile industry and evaluated mobile development platforms on his tech blogs and is well-known within the blogosphere. Shane received a B.S. in Computer Science from the University of California.

A self-admitted gadget freak, Shane always has the latest smartphone, tablet, or other mobile device. He can often be found fiddling with the latest technologies, such as cloud services and mobile platforms, and other exciting, state-of-the-art technologies that activate the creative part of his brain. He also enjoys traveling the world with his geeky wife, even if she did make him dive with 4-meter-long great white sharks and almost get eaten by a lion in Kenya. He admits that he has to take at least two phones with him when backpacking—even though there is no coverage—and that he snickered and whipped out his Android phone to take a picture when Laurie got her tongue stuck to that iceberg in Iceland, and that he is catching on that he should be writing his own bio.

Introduction

Pioneered by the Open Handset Alliance and Google, Android is a popular, free, open-source mobile platform that has taken the wireless world by storm. This book, and the next volume, *Android Wireless Application Development Volume II: Advanced Topics,* provide comprehensive guidance for software development teams on designing, developing, testing, debugging, and distributing professional Android applications. If you're a veteran mobile developer, you can find tips and tricks to streamline the development process and take advantage of Android's unique features. If you're new to mobile development, these books provide everything you need to make a smooth transition from traditional software development to mobile development—specifically, its most promising platform: Android.

Who Should Read This Book

This book include tips for successful mobile development based upon our years in the mobile industry and covers everything you need to know in order to run a successful Android project from concept to completion. We cover how the mobile software process differs from traditional software development, including tricks to save valuable time and pitfalls to avoid. Regardless of the size of your project, this book is for you.

This book was written for several audiences:

- **Software developers who want to learn to develop professional Android applications.** The bulk of this book is targeted at software developers with Java experience who do not necessarily have mobile development experience. More seasoned developers of mobile applications can learn how to take advantage of Android and how it differs from the other technologies of the mobile development market today.
- **Quality assurance personnel tasked with testing Android applications.** Whether they are black-box or white-box testing, quality assurance engineers can find this book invaluable. We devote several chapters to mobile QA concerns, including topics such as developing solid test plans and defect-tracking systems for mobile applications, how to manage handsets, and how to test applications thoroughly using all the Android tools available.
- **Project managers planning and managing Android development teams.** Managers can use this book to help plan, hire, and execute Android projects from start to finish. We cover project risk management and how to keep Android projects running smoothly.

- **Other audiences.** This book is useful not only to the software developer, but also to the corporation looking at potential vertical market applications, the entrepreneur thinking about a cool phone application, and the hobbyist looking for some fun with his or her new phone. Businesses seeking to evaluate Android for their specific needs (including feasibility analysis) can also find the information provided valuable. Anyone with an Android handset and a good idea for a mobile application can put the information provided in this book to use for fun and profit.

Key Questions Answered in This Volume

This volume of the book answers the following questions:

1. What is Android? How do the SDK versions differ?
2. How is Android different from other mobile technologies, and how can developers take advantage of these differences?
3. How do developers use the Eclipse Development Environment for Java to develop and debug Android applications on the emulator and handsets?
4. How are Android applications structured?
5. How do developers design robust user interfaces for mobile—specifically, for Android?
6. What capabilities does the Android SDK have and how can developers use them?
7. How does the mobile development process differ from traditional desktop development?
8. What development strategies work best for Android development?
9. What do managers, developers, and testers need to look for when planning, developing, and testing a mobile development application?
10. How do mobile teams design bulletproof Android applications for publication?
11. How do mobile teams package Android applications for deployment?
12. How do mobile teams make money from Android applications?
13. And, finally, what is new in this edition of the book?

How These Books Are Structured

We wrote the first edition of this book before the Android SDK was released. Now, three years and 14 Android SDK releases later, there is so much to talk about that we've had to divide the content of *Android Wireless Application Development* into two separate volumes for this, the third edition.

Android Wireless Application Development Volume I: Android Essentials focuses on Android essentials, including setting up your development environment, understanding the application lifecycle, user interface design, developing for different types of devices, and the mobile software process from design and development to testing and publication of commercial-grade applications.

Android Wireless Application Development Volume II: Advanced Topics focuses on advanced Android topics, including leveraging various Android APIs for threading, networking, location-based services, hardware sensors, animation, graphics, and more. Coverage of advanced Android application components, such as services, application databases, content providers, and intents, is also included. Developers learn to design advanced user interface components and integrate their applications deeply into the platform. Finally, developers learn how to extend their applications beyond traditional boundaries using optional features of the Android platform, including the Android Native Development Kit (NDK), Cloud-To-Device Messaging service (C2DM), Android Market In-Application Billing APIs, Google Analytics APIs, and more.

Android Wireless Application Development Volume I: Android Essentials is divided into six parts. The first four parts are primarily of interest to developers; Part V provides lots of helpful information for project managers and quality assurance personnel as well as developers. Part VI includes several helpful appendixes to help you get up and running with the most important Android tools.

Here is an overview of the various parts in this book:

- **Part I: An Overview of the Android Platform**
  Part I provides an introduction to Android, explaining how it differs from other mobile platforms. You become familiar with the Android SDK and tools, install the development tools, and write and run your first Android application—on the emulator and on a handset.
- **Part II: Android Application Basics**
  Part II introduces the design principles necessary to write Android applications. You learn how Android applications are structured and how to include resources, such as strings, graphics, and user interface components, in your projects.
- **Part III: Android User Interface Design Essentials**
  Part III dives deeper into how user interfaces are designed in Android. You learn about the core user interface element in Android: the `View`. You also learn about the most common user interface controls and layouts provided in the Android SDK.
- **Part IV: Android Application Design Essentials**
  Part IV covers the features used by most Android applications, including storing persistent application data using preferences and working with files, directories, and content providers. You also learn how to design applications that will run smoothly on many different Android devices.

- **Part V: Publishing and Distributing Android Applications**
  Part V covers the software development process for mobile, from start to finish, with tips and tricks for project management, software developers, and quality assurance personnel.
- **Part VI: Appendixes**
  Part VI includes two helpful quick-start guides for the Android development tools—the emulator and DDMS—as well as an appendix of Eclipse tips and tricks.

An Overview of Changes in This Edition

When we began writing the first edition of this book, there were no Android devices on the market. One Android device became available shortly after we started writing, and it was available only in the United States. Today there are hundreds of devices shipping all over the world—smartphones, tablets, e-book readers, and specialty devices such as the Google TV. The Android platform has gone through extensive changes since the first edition of this book was published. The Android SDK has many new features and the development tools have received many much-needed upgrades. Android, as a technology, is now on solid footing within the mobile marketplace.

In this new edition, we took the opportunity to do a serious overhaul of the book content. But don't worry, it's still the book readers loved the first (and second!) time; it's just bigger, better, and more comprehensive. In order to cover more of the exciting topics available to Android developers, we had to divide the book into two volumes. In addition to adding tons of new content, we've retested and upgraded all existing content (text and sample code) for use with the latest Android SDKs available while still remaining backward compatible. The Android development community is diverse, and we aim to support all developers, regardless of which devices they are developing for. This includes developers who need to target nearly all platforms, so coverage in some key areas of older SDKs continues to be included because it's often the most reasonable option for compatibility.

Here are some of the highlights of the additions and enhancements we've made to this edition:

- Coverage of the latest and greatest Android tools and utilities.
- Updates to all existing chapters, often with some entirely new sections.
- New chapters, which cover new SDK features or expand upon those covered in previous editions.
- Updated sample code and applications, conveniently organized by chapter.
- Topics such as Android manifest files, content providers, designing apps, and testing now have their own chapters.

- Coverage of hot topics such as application compatibility, designing for different devices, and working with relatively new user interface components such as fragments.
- Even more tips and tricks from the trenches to help you design, develop, and test applications for different device targets, including an all-new chapter on tackling compatibility issues.

As you can see, we cover many of the hottest and most exciting features that Android has to offer. We didn't take this review lightly; we touched every existing chapter, updated content, and added many new chapters as well. Finally, we included many additions, clarifications, and, yes, even a few fixes based on the feedback from our fantastic (and meticulous) readers. Thank you!

Development Environment Used in This Book

The Android code in this book was written using the following development environments:

- Windows 7 and Mac OS X 10.7.x
- Eclipse Java IDE Version 3.7 (Indigo) and Version 3.6 (Helios)
- Eclipse JDT plug-in and Web Tools Platform (WTP)
- Java SE Development Kit (JDK) 6 Update 26
- Android SDK Version 2.3.4, API Level 10 (Gingerbread MR1); Android SDK Version 3.2, API Level 13 (Honeycomb MR2); Android SDK Version 4.0, API Level 14 (Ice Cream Sandwich)
    1. ADT Plug-in for Eclipse 15.0.0
    2. SDK Tools Revision 15

  Android Devices: Samsung Nexus S, HTC Evo 4G, Motorola Droid 3, Samsung Galaxy Tab 10.1, Motorola Xoom, Motorola Atrix 4G, and Sony Ericsson Xperia Play

The Android platform continues to aggressively grow in market share against competing mobile platforms, such as Apple iOS and BlackBerry. New and exciting types of devices reach consumers' hands at a furious pace, with new editions of the Android platform appearing all the time. Developers can no longer ignore Android as a target platform if they want to reach the smartphone (or smart-device) users of today and tomorrow.

Android's latest major platform update, Android 4.0—frequently called by its codename, Ice Cream Sandwich, or just ICS—merges the smartphone-centric Android 2.3.x (Gingerbread) and the tablet-centric Android 3.x (Honeycomb) platform editions into a single SDK for all smart-devices, be they phones, tablets, televisions, or toasters. This

book features the latest SDK and tools available, but it does not focus on them to the detriment of popular legacy versions of the platform. This book is meant to be an overall reference to help developers support all popular devices on the market today. As of the writing of this book, only a very small percentage (less than 5%) of users' devices are running Android 3.0 or 4.0. Of course, some devices will receive upgrades, and users will purchase new Ice Cream Sandwich devices as they become available, but for now, developers need to straddle this gap and support numerous versions of Android to reach the majority of users in the field.

So what does this mean for this book? It means we provide both legacy API support and discuss some of the newer APIs only available in later versions of the Android SDK. We discuss strategies for supporting all (or at least most) users in terms of compatibility. And we provide screenshots that highlight different versions of the Android SDK, because each major revision has brought with it a change in the look and feel of the overall platform. That said, we are assuming that you are downloading the latest Android tools, so we provide screenshots and steps that support the latest tools available at the time of writing, not legacy tools. Those are the boundaries we set when trying to determine what to include or leave out of this book.

Supplementary Materials Available

The source code that accompanies this book is available for download on the publisher website: www.informit.com/title/9780321813831. The source code is also available for download from our book website: http://androidbook.blogspot.com/p/book-code-downloads.html (http://goo.gl/kyAsN). You'll also find a variety of Android topics discussed at our book website (http://androidbook.blogspot.com). For example, we present reader feedback, questions, and further information. You can find links to our various technical articles on our book website as well.

Where to Find More Information

There is a vibrant, helpful Android developer community on the Web. Here are a number of useful websites for Android developers and followers of the wireless industry:

- **Android Developer website:** The Android SDK and developer reference site: http://developer.android.com or http://d.android.com
- **Stack Overflow:** The Android website with great technical information (complete with tags) and an official support forum for developers: http://stackoverflow.com/questions/tagged/android
- **Open Handset Alliance:** Android manufacturers, operators, and developers: http://www.openhandsetalliance.com

- **Android Market:** Buy and sell Android applications:
  http://www.android.com/market
- **Mobiletuts+:** Mobile development tutorials, including Android:
  http://mobile.tutsplus.com/category/tutorials/android
- **anddev.org:** An Android developer forum:
  http://www.anddev.org
- **Google Team Android Apps:** Open-source Android applications:
  http://apps-for-android.googlecode.com
- **Android Tools Project Site:** The tools team discusses updates and changes:
  https://sites.google.com/a/android.com/tools/recent
- **FierceDeveloper:** A weekly newsletter for wireless developers:
  http://www.fiercedeveloper.com
- **Wireless Developer Network:** Daily news on the wireless industry:
  http://www.wirelessdevnet.com
- **XDA-Developers Android Forum**: From general development to ROMs:
  http://forum.xda-developers.com/forumdisplay.php?f=564
- **Developer.com:** A developer-oriented site with mobile articles:
  http://www.developer.com

Conventions Used in This Book

This book uses the following conventions:

- ➥ is used to signify to readers that the authors meant for the continued code to appear on the same line. No indenting should be done on the continued line.
- Code or programming terms are set in `monospace` text.
- Java import statements, exception handling, and error checking are often removed from printed code examples for clarity and to keep the book a reasonable length.

This book also presents information in the following sidebars:

Tip

Tips provide useful information or hints related to the current text.

Note

Notes provide additional information that might be interesting or relevant.

Warning

Warnings provide hints or tips about pitfalls that may be encountered and how to avoid them.

Contacting the Authors

We welcome your comments, questions, and feedback. We invite you to visit our blog at:

- http://androidbook.blogspot.com

Or email us at:

- androidwirelessdev+awad3ev1@gmail.com

Circle us on Google+:

- Lauren Darcey: http://goo.gl/P3RGo
- Shane Conder: http://goo.gl/BpVJh

3

Writing Your First Android Application

You should now have a workable Android development environment set up on your computer. Hopefully, you also have an Android device as well. Now it's time for you to start writing some Android code. In this chapter, you learn how to add and create Android projects in Eclipse and verify that your Android development environment is set up correctly. You also write and debug your first Android application in the software emulator and on an Android device.

Note

The Android development tools are updated frequently. We have made every attempt to provide the latest steps for the latest tools. However, these steps and the user interfaces described in this chapter may change at any time. Please review the Android development website (http://d.android.com/sdk) and our book website (http://androidbook.blogspot.com) for the latest information.

Testing Your Development Environment

The best way to make sure you configured your development environment correctly is to take an existing Android application and run it. You can do this easily by using one of the sample applications provided as part of the Android SDK in the `samples` subdirectory found where your Android SDK is installed.

Within the Android SDK sample applications, you will find a classic game called *Snake* (http://goo.gl/wRojX). To build and run the Snake application, you must create a new Android project in your Eclipse workspace based on the existing Android sample project, create an appropriate Android Virtual Device (AVD) profile, and configure a launch configuration for that project. After you have everything set up correctly, you can build the application and run it on the Android emulator and on an Android device. By testing your development environment with a sample application, you can rule out project configuration and coding issues and focus on determining whether the tools are set

up properly for Android development. After this fact has been established, you can move on to writing and compiling your own applications.

Adding the Snake Project to Your Eclipse Workspace

The first thing you need to do is add the Snake project to your Eclipse workspace. To do this, follow these steps:

1. Choose File, New, Other….
2. Choose Android, Android Sample Project (see Figure 3.1). Click Next.

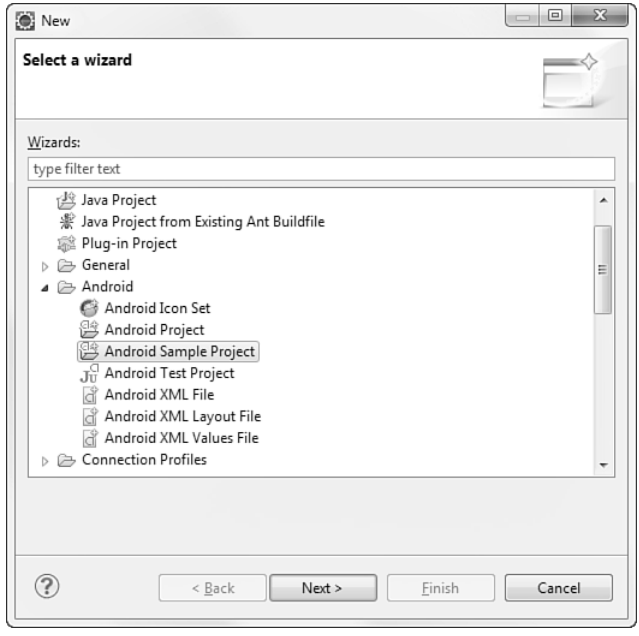

Figure 3.1 Creating a new Android project.

3. Choose your build target (see Figure 3.2). In this case, we've picked Android 4.0, API Level 14, from the Android Open Source Project. Click Next.
4. Next, select which sample you want to create (see Figure 3.3). Choose Snake.
5. Click Finish. You now see the Snake project files in your workspace (see Figure 3.4).

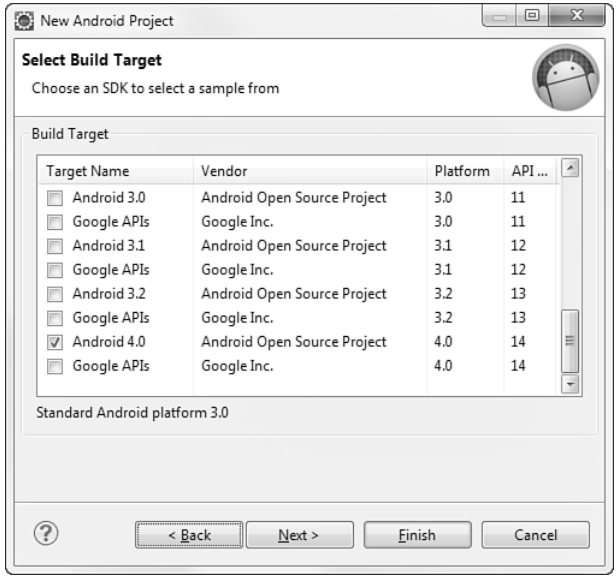

Figure 3.2 Choose an API level for the sample.

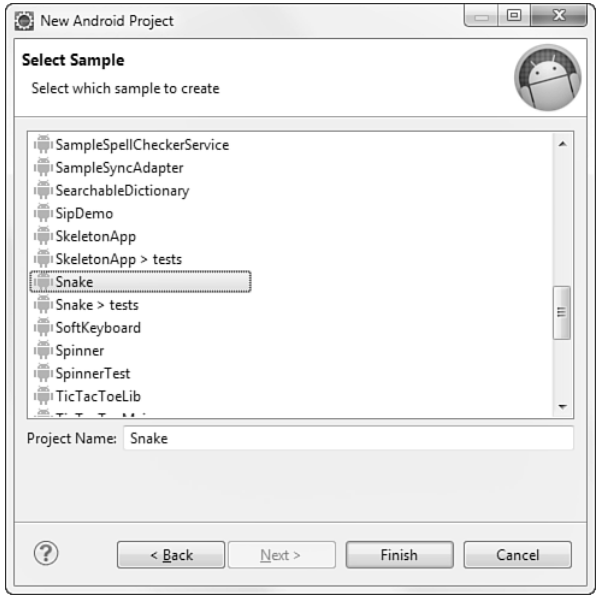

Figure 3.3 Picking the sample project.

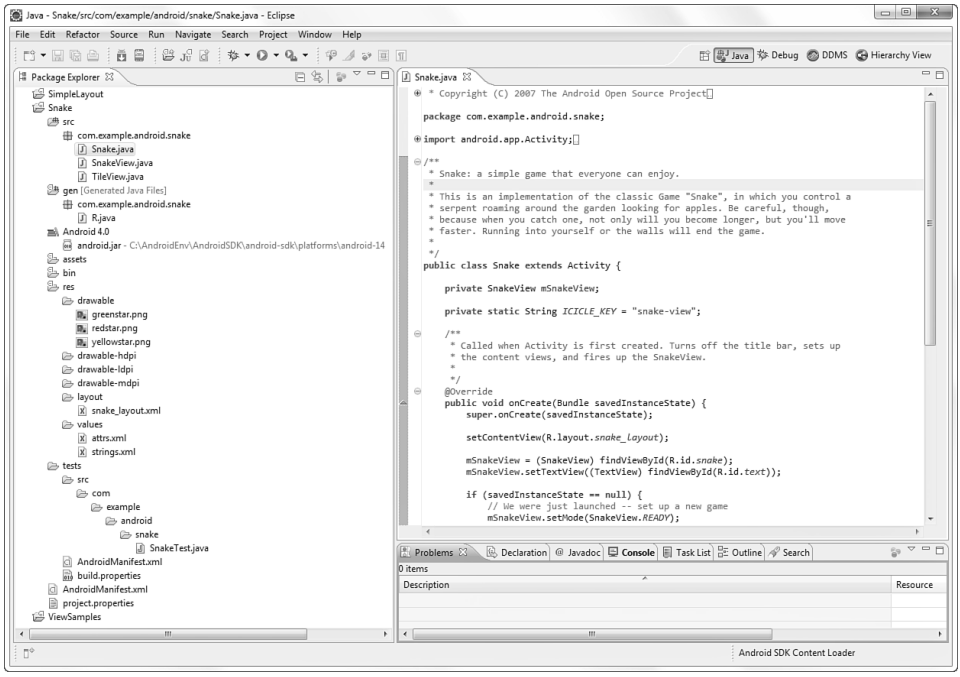

Figure 3.4 The Snake project files.

Warning

Occasionally Eclipse shows the error "Project 'Snake' is missing required source folder: gen" when you're adding an existing project to the workspace. If this happens, navigate to the project file called `R.java` under the `/gen` directory and delete it. The `R.java` file is automatically regenerated and the error should disappear. Performing a Clean operation followed by a Build operation does not always solve this problem.

Creating an Android Virtual Device (AVD) for Your Snake Project

The next step is to create an AVD that describes what type of device you want to emulate when running the Snake application. This AVD profile describes what type of device you want the emulator to simulate, including which Android platform to support. You do not need to create new AVDs for each application, only for each device you want to emulate. You can specify different screen sizes and orientations, and you can specify whether the emulator has an SD card and, if it does, what capacity the card is.

For the purposes of this example, an AVD for the default installation of Android 2.3.3 suffices. Here are the steps to create a basic AVD:

1. Launch the Android Virtual Device Manager from within Eclipse by clicking the little Android device icon on the toolbar (). If you cannot find the icon, you can also launch the manager through the Window menu of Eclipse.
2. Click the New button.
3. Choose a name for your AVD. Because we are going to take all the defaults, give this AVD a name of `Android_Vanilla4.0`.
4. Choose a build target. We want a typical Android 4.0 device, so choose Google APIs (Google Inc.) – API Level 14 from the drop-down menu. This will include the Google Android applications, such as the Maps application, as part of the platform image.
5. Choose an SD card capacity, in either kibibytes or mibibytes. Not familiar with kibibytes? See this Wikipedia entry: http://goo.gl/N3Rdd. This SD card image will take up space on your hard drive, so choose something reasonable, such as 1024MiB.
6. Seriously consider enabling the Snapshot feature. This feature greatly improves emulator startup performance. See Appendix A, "The Android Emulator Quick-Start Guide," for details.

Warning

As of this writing, there's a known issue with the Android Tools R14 emulator that prevents the snapshot feature from working correctly. See http://goo.gl/pnMt0 for more information on the current known issues.

7. Choose a skin. This option controls the different resolutions of the emulator. In this case, we use the recommended WVGA800 screen. (The emulator in Tools R14 has known performance issues with running the standard Android 4.0 screen resolution of WXGA720.) This skin most directly correlates to the popular 4.0 devices. Feel free to choose the most appropriate skin to match the Android device on which you plan to run the application.

   Your project settings will look like Figure 3.5.
8. Click the Create AVD button and then wait for the operation to complete.
9. Click Finish. Because the AVD manager formats the memory allocated for SD card images, creating AVDs with SD cards could take a few moments.

For more information on creating different types of AVDs, check out Appendix A.

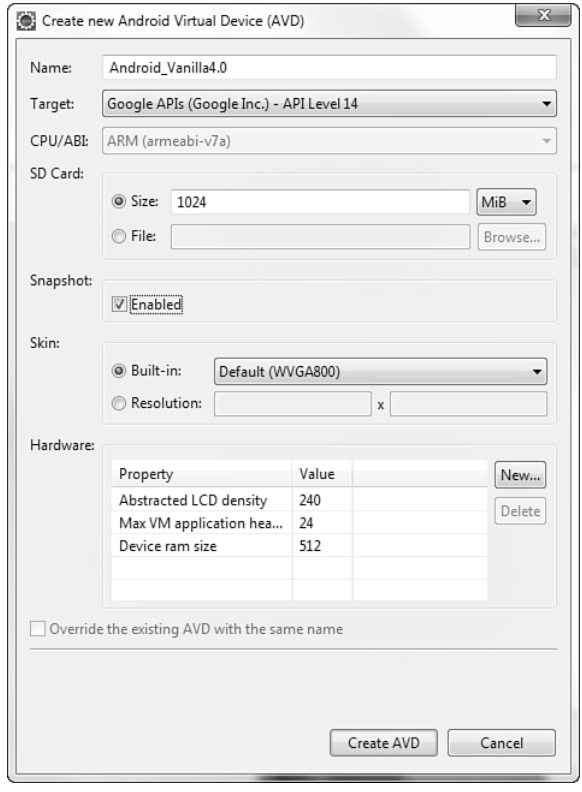

Figure 3.5 Creating a new AVD.

Creating a Launch Configuration for Your Snake Project

Next, you must create a launch configuration in Eclipse to configure under what circumstances the Snake application builds and launches. The launch configuration is where you configure the emulator options to use and the entry point for your application.

You can create Run configurations and Debug configurations separately, each with different options. These configurations are created under the Run menu in Eclipse (Run, Run Configurations and Run, Debug Configurations). Follow these steps to create a basic Debug configuration for the Snake application:

1. Choose Run, Debug Configurations.
2. Double-click Android Application to create a new configuration.
3. Name your Debug configuration `SnakeDebugConfiguration`.
4. Choose the project by clicking the Browse button and choosing the Snake project.

5. Switch to the Target tab and, from the preferred AVD list, choose the `Android_Vanilla4.0` AVD created earlier, as shown in Figure 3.6.

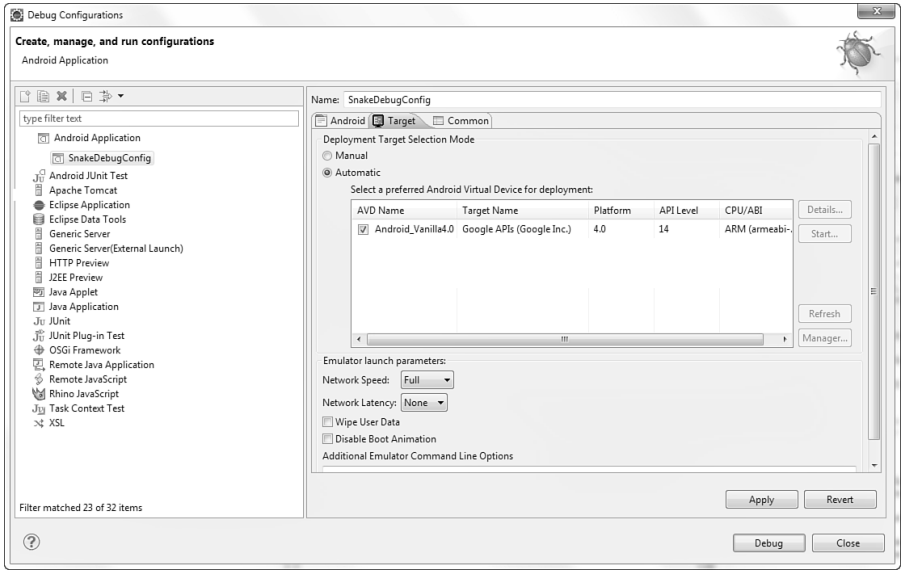

Figure 3.6 The Snake application Debug configuration in Eclipse.

You can set other emulator and launch options on the Target and Common tabs, but for now we are leaving the defaults as they are.

Running the Snake Application in the Android Emulator

Now you can run the Snake application using the following steps:

1. Choose the Debug As icon drop-down menu on the toolbar ( ).
2. Pull the drop-down menu and choose the `SnakeDebugConfiguration` you created. If you do not see your new configuration listed, find it in the Debug Configurations listing and click the Debug button. Subsequent launches can be initiated from the little bug drop-down.
3. The Android emulator starts up; this might take a few moments to initialize. Then the application will be installed or reinstalled onto the emulator.

Tip

It can take a long time for the emulator to start up, even on very fast computers. You might want to leave it around while you work and reattach to it as needed. The tools in Eclipse handle reinstalling the application and re-launching the application, so you can more easily keep the emulator loaded all the time. This is another reason to enable the Snapshot feature for each AVD. You can also use the Start button on the Android Virtual Device Manager to load up an emulator before you need it. Launching the AVD this way also gives you some additional options such as screen scaling (see Figure 3.7), which can be used to either fit the AVD on your screen if it's very high resolution or more closely emulate the size it might be on real hardware.

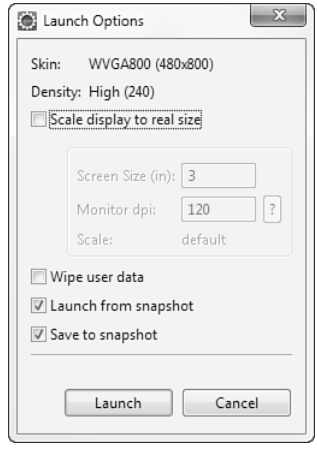

Figure 3.7 Configuring AVD launch options.

4. If necessary, swipe the screen from left to right to unlock the emulator, as shown in Figure 3.8.
5. The Snake application starts and you can play the game, as shown in Figure 3.9.

You can interact with the Snake application through the emulator and play the game. You can also launch the Snake application from the Application drawer at any time by clicking its application icon. There is no need to shut down and restart the emulator every time you rebuild and reinstall your application for testing. Simply leave the emulator running on your computer in the background while you work in Eclipse and then redeploy using the Debug configuration again.

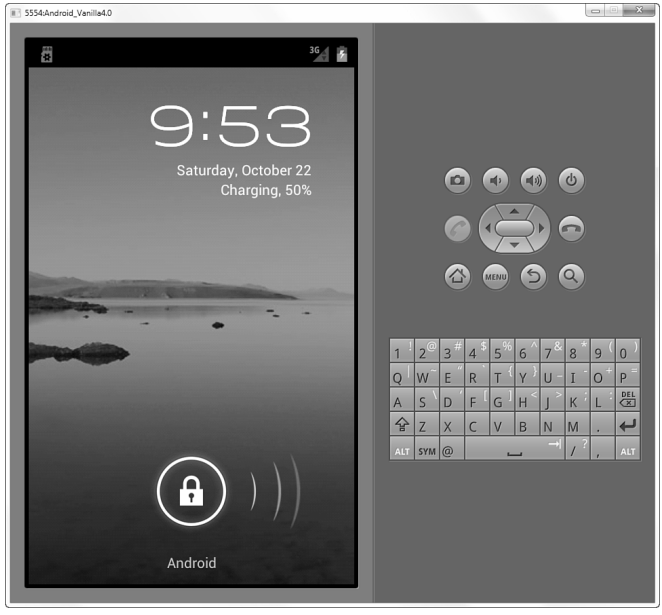

Figure 3.8 The Android emulator launching (locked).

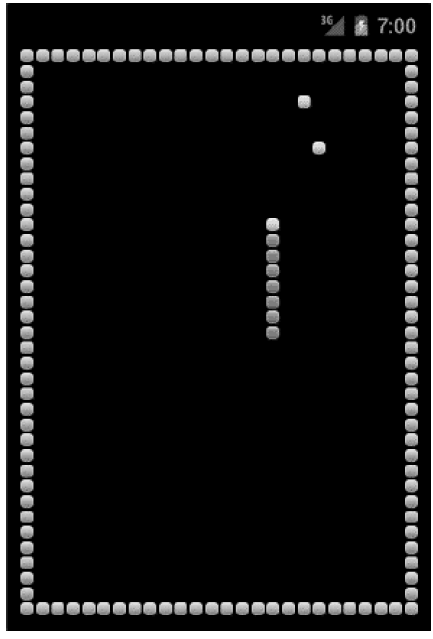

Figure 3.9 The Snake game in the Android emulator.

Building Your First Android Application

Now it's time to write your first Android application from scratch. To get your feet wet, you will start with a simple "Hello World" application and build upon it to explore some of the features of the Android platform in more detail.

Tip

The code examples provided in this chapter are taken from the MyFirstAndroidApp application. The source code for the MyFirstAndroidApp application is provided for download on the book's websites.

Creating and Configuring a New Android Project

You can create a new Android application in much the same way as when you added the Snake application to your Eclipse workspace.

The first thing you need to do is create a new project in your Eclipse workspace. The Android Project Wizard creates all the required files for an Android application. Follow these steps within Eclipse to create a new project:

1. Choose File, New, Android Project, or choose the Android Project creator icon, which looks like a folder (), on the Eclipse toolbar.
2. Choose a project name, as shown in Figure 3.10. In this case, name the project `MyFirstAndroidApp`.

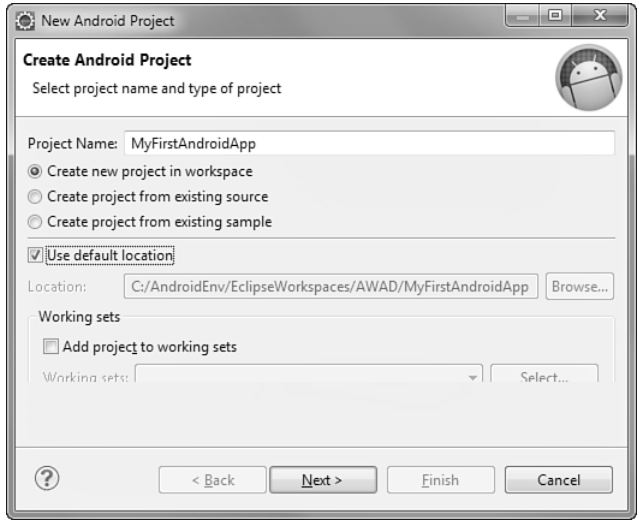

Figure 3.10 Configuring a new Android project.

3. Choose a location for the project files. Because this is a new project, select the Create New Project in Workspace radio button. Check the Use Default Location check box or change the directory to wherever you want to store the source files. Click Next.
4. Select a build target for your application, as shown in Figure 3.11. Choose a target that is compatible with the Android devices you have in your possession. For this example, you might use the Android 2.3.3 target or, for Ice Cream Sandwich devices, Android 4.0 (API Level 14). Click Next.

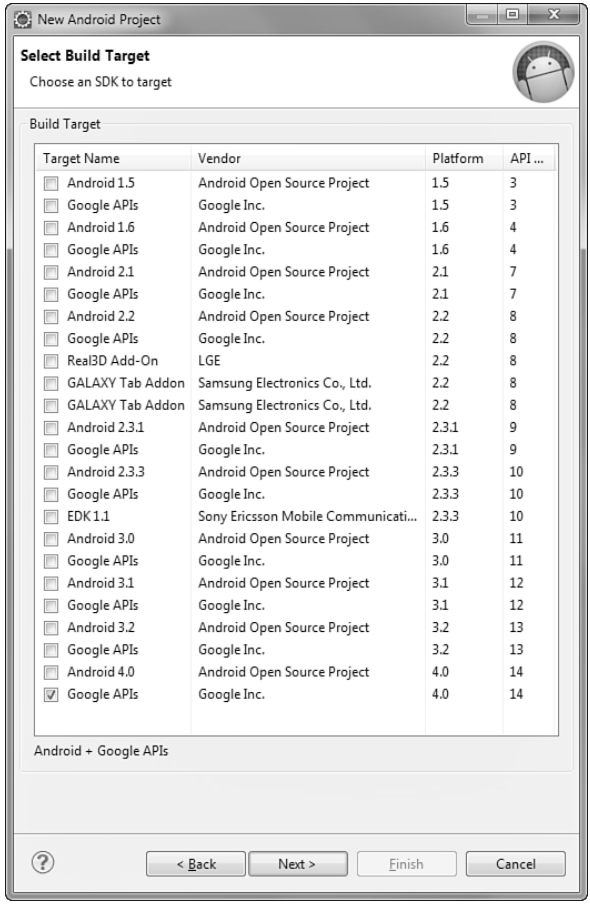

Figure 3.11 Choosing a build target for a new Android project.

5. Configure your application information. Choose an application name. The application name is the "friendly" name of the application and the name shown with the icon on the application launcher. In this case, the application name is "My First Android App."
6. Choose a package name. Here you should follow standard package namespace conventions for Java. Because all our code examples in this book fall under the `com.androidbook.*` namespace, we will use the package name `com.androidbook.myfirstandroidapp`, but you are free to choose your own package name.
7. Check the Create Activity check box. This instructs the wizard to create a default launch activity for the application. Call this `Activity` class `MyFirstAndroidAppActivity`.
8. Set the minimum SDK version. This value should be the same or lower than the target SDK API level. Because our application will be compatible with just about any Android device, you can set this number low (like to 4 to represent Android 1.6) or at the target API level to avoid annoying warnings in Eclipse. Make sure you set the minimum SDK version to encompass any test devices you have available so you can successfully install the application on them.

   Your project settings should look like Figure 3.12.
9. Finally, click the Finish button.

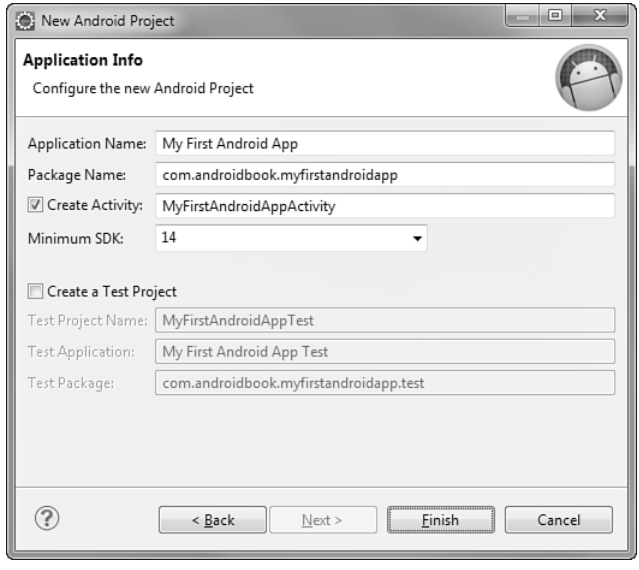

Figure 3.12 Configuring My First Android App using the Android Project Wizard.

Core Files and Directories of the Android Application

Every Android application has a set of core files that are created and used to define the functionality of the application. The following files are created by default with a new Android application:

- **`AndroidManifest.xml`**—The central configuration file for the application. It defines your application's capabilities and permissions as well as how it runs.
- **`project.properties`**—A generated build file used by Eclipse and the Android ADT plug-in. It defines your application's build target and other build system options, as required. Do not edit this file.
- **`proguard.cfg`**—A generated build file used by Eclipse, ProGuard, and the Android ADT plug-in. Edit this file to configure your code optimization and obfuscation settings for release builds.
- **`/src` folder**—Required folder for all source code.
- **`/src/com/androidbook/myfirstandroidapp/MyFirstAndroidApp Activity.java`**—Main entry point to this application, named `MyFirstAndroidAppActivity`. This activity has been defined as the default launch activity in the Android manifest file.
- **`/gen/com/androidbook/myfirstandroidapp/R.java`**—A generated resource management source file. Do not edit this file.
- **`/assets` folder**—Required folder where uncompiled file resources can be included in the project. Application assets are pieces of application data (files, directories) that you do not want managed as application resources.
- **`/res` folder**—Required folder where all application resources are managed. Application resources include animations, drawable graphics, layout files, data-like strings and numbers, and raw files.
- **`/res/drawable-*`**—Application icon graphic resources are included in several sizes for different device screen resolutions.
- **`/res/layout/main.xml`**—Layout resource file used by `MyFirstAndroidAppActivity` to organize controls on the main application screen.
- **`/res/values/strings.xml`**—The resource file where string resources are defined.

A number of other files are saved on disk as part of the Eclipse project in the workspace. However, the files and resource directories included in the list here are the important project files you will use on a regular basis.

Creating an AVD for Your Project

The next step is to create an AVD that describes what type of device you want to emulate when running the application. For this example, we can use the AVD we created for

the Snake application. An AVD describes a device, not an application. Therefore, you can use the same AVD for multiple applications. You can also create similar AVDs with the same configuration but different data (such as different applications installed and different SD card contents).

Creating a Launch Configuration for Your Project

Next, you must create a Run and Debug launch configuration in Eclipse to configure the circumstances under which the MyFirstAndroidApp application builds and launches. The launch configuration is where you configure the emulator options to use and the entry point for your application.

You can create Run configurations and Debug configurations separately, with different options for each. Begin by creating a Run configuration for the application. Follow these steps to create a basic Run configuration for the MyFirstAndroidApp application:

1. Choose Run, Run Configurations (or right-click the project and choose Run As).
2. Double-click Android Application.
3. Name your Run configuration `MyFirstAndroidAppRunConfig`.
4. Choose the project by clicking the Browse button and choosing the MyFirstAndroidApp project.
5. Switch to the Target tab and set the Device Target Selection Mode to Manual.

Tip

If you leave the Device Target Selection Mode set to Automatic when you choose Run or Debug in Eclipse, your application is automatically installed and run on the device if the device is plugged in. Otherwise, the application starts in the emulator with the specified AVD. By choosing Manual, you are always prompted for whether (a) you want your application to be launched in an existing emulator; (b) you want your application to be launched in a new emulator instance and are allowed to specify an AVD; or (c) you want your application to be launched on the device (if it's plugged in). If any emulator is already running, the device is then plugged in, and the mode is set to Automatic, you see this same prompt, too.

Now create a Debug configuration for the application. This process is similar to creating a Run configuration. Follow these steps to create a basic Debug configuration for the MyFirstAndroidApp application:

1. Choose Run, Debug Configurations (or right-click the project and choose Debug As).
2. Double-click Android Application.
3. Name your Debug configuration `MyFirstAndroidAppDebugConfig`.

4. Choose the project by clicking the Browse button and choosing the MyFirstAndroidApp project.
5. Switch to the Target tab and set the Device Target Selection Mode to Manual.
6. Click Apply and then click Close.

You now have a Debug configuration for your application.

Running Your Android Application in the Emulator

Now you can run the MyFirstAndroidApp application using the following steps:

1. Choose the Run As icon drop-down menu on the toolbar ( ).
2. Pull the drop-down menu and choose the Run configuration you created. (If you do not see it listed, choose the Run Configurations... item and select the appropriate configuration. The Run configuration shows up on this drop-down list the next time you run the configuration.)
3. Because you chose the Manual Target Selection mode, you are now prompted for your emulator instance. Change the selection to Launch a New Android Virtual Device and then select the AVD you created, as shown in Figure 3.13. Here you can choose from an already-running emulator or launch a new instance with an AVD that is compatible with the application settings.

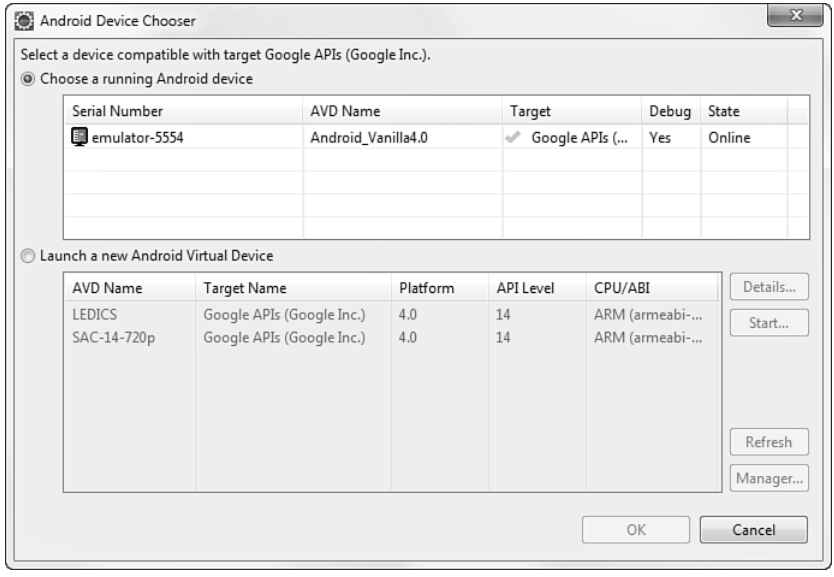

Figure 3.13 Manually choosing a target selection mode.

4. The Android emulator starts up, which might take a moment.
5. Click the Menu button or push the slider to the right to unlock the emulator.
6. The application starts, as shown in Figure 3.14.

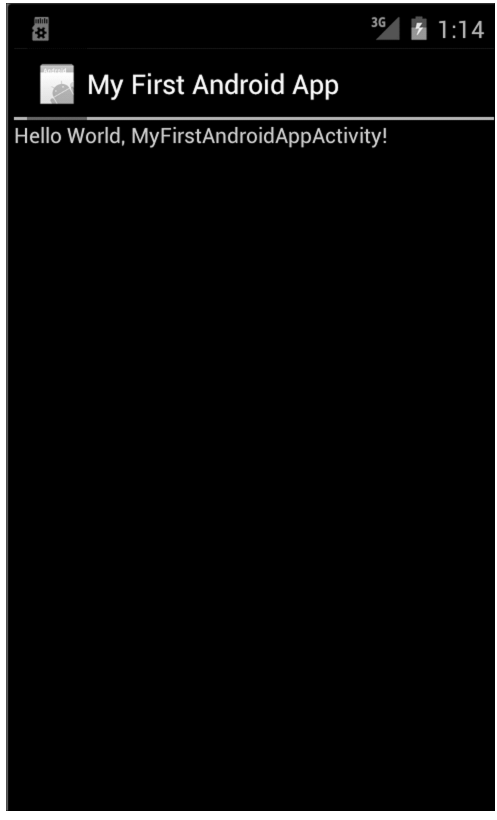

Figure 3.14 My First Android App running in the emulator.

7. Click the Back button in the Emulator to end the game or click Home to suspend it.
8. Click the grid button to browse all installed applications. Your screen looks something like Figure 3.15.
9. Click the My First Android Application icon to launch the application again.

Figure 3.15 The My First Android App icon shown in the application listing.

Debugging Your Android Application in the Emulator

Before we go any further, you need to become familiar with debugging in the emulator. To illustrate some useful debugging tools, let's manufacture an error in the My First Android Application.

In your project, edit the source file called `MyFirstAndroidApp.java`. Create a new method called `forceError()` in your class and make a call to this method in your `Activity` class's `onCreate()` method. The `forceError()` method forces a new unhandled error in your application.

The `forceError()` method should look something like this:

```
public void forceError() {
    if(true) {
        throw new Error("Whoops");
    }
}
```

It's probably helpful at this point to run the application and watch what happens. Do this using the Run configuration first. In the emulator, you see that the application has stopped unexpectedly. You are prompted by a dialog that enables you to forcefully close the application, as shown in Figure 3.16.

Figure 3.16 My First Android App crashing gracefully.

Shut down the application but keep the emulator running. Now it's time to debug. You can debug the MyFirstAndroidApp application using the following steps:

1. Choose the Debug As icon drop-down menu on the toolbar.
2. Pull the drop-down menu and choose the Debug configuration you created. (If you do not see it listed, choose the Debug Configurations... item and select the

appropriate configuration. The Debug configuration shows up on this drop-down list the next time you run the configuration.)

3. Continue as you did with the Run configuration and choose the appropriate AVD and then launch the emulator again, unlocking it if needed.

It takes a moment for the debugger to attach. If this is the first time you've debugged an Android application, you may need to click through some dialogs, such as the one shown in Figure 3.17, the first time your application attaches to the debugger.

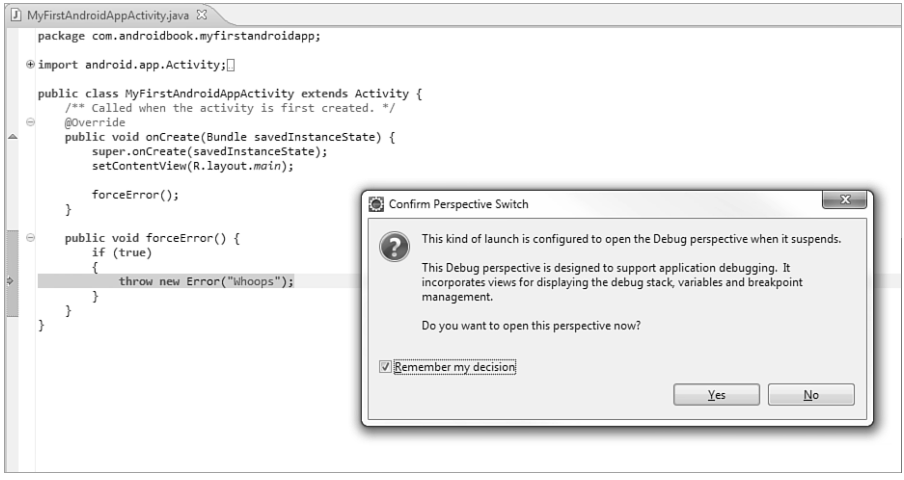

Figure 3.17 Switching debug perspectives for Android emulator debugging.

In Eclipse, use the Debug perspective to set breakpoints, step through code, and watch the LogCat logging information about your application. This time, when the application fails, you can determine the cause using the debugger. You might need to click through several dialogs as you set up to debug within Eclipse. If you allow the application to continue after throwing the exception, you can examine the results in the Debug perspective of Eclipse. If you examine the LogCat logging pane, you see that your application was forced to exit due to an unhandled exception (see Figure 3.18).

Specifically, there's a red `AndroidRuntime` error: `java.lang.Error: Whoops`. Back in the emulator, click the Force Close button. Now set a breakpoint on the `forceError()` method by right-clicking the left side of the line of code and choosing Toggle Breakpoint (or pressing Ctrl+Shift+B).

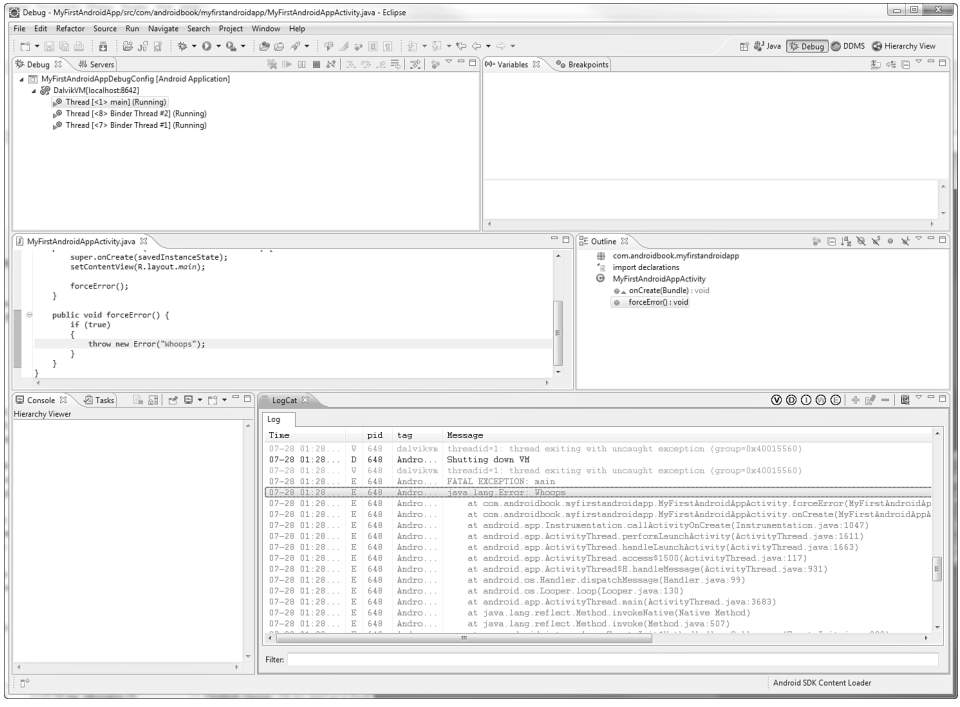

Figure 3.18 Debugging My First Android App in Eclipse.

Tip

In Eclipse, you can step through code using Step Into (F5), Step Over (F6), Step Return (F7), or Resume (F8). On Mac OS X, you might find that the F8 key is mapped globally. If you want to use the keyboard convenience command, you might want to change the keyboard mapping in Eclipse by choosing Eclipse, Preferences, General, Keys and then finding the entry for Resume and changing it to something else. Alternatively, you can change the Mac OS X global mapping by going to System Preferences, Keyboard & Mouse, Keyboard Shortcuts and then changing the mapping for F8 to something else.

In the emulator, restart your application and step through your code. You see that your application has thrown the exception and then the exception shows up in the Variable Browser pane of the Debug Perspective. Expanding its contents shows that it is the "Whoops" error.

This is a great time to crash your application repeatedly and get used to the controls. While you're at it, switch over to the DDMS perspective. You note the emulator has a list of processes running on the device, such as `system_process` and `com.android.phone`. If you launch `MyFirstAndroidApp`, you see `com.androidbook.myfirstandroidapp` show up as a process on the emulator listing. Force the app

to close because it crashes, and note that it disappears from the process list. You can use DDMS to kill processes, inspect threads and the heap, and access the phone file system.

Adding Logging Support to Your Android Application

Before you start diving into the various features of the Android SDK, you should familiarize yourself with logging, a valuable resource for debugging and learning Android. Android logging features are in the `Log` class of the `android.util` package. Some helpful methods in the `android.util.Log` class are shown in Table 3.1.

Table 3.1 **Commonly Used Logging Methods**

Method	Purpose
`Log.e()`	Log errors
`Log.w()`	Log warnings
`Log.i()`	Log informational messages
`Log.d()`	Log debug messages
`Log.v()`	Log verbose messages

To add logging support to `MyFirstAndroidApp`, edit the file `MyFirstAndroidApp.java`. First, you must add the appropriate import statement for the `Log` class:

```
import android.util.Log;
```

Tip

To save time in Eclipse, you can use the imported classes in your code and add the imports needed by hovering over the imported class name and choosing the Add Imported Class option.

You can also use the Organize Imports command (Ctrl+Shift+O in Windows or Command+Shift+O on a Mac) to have Eclipse automatically organize your imports. This removes unused imports and adds new ones for packages used but not imported. If a naming conflict arises, as it often does with the `Log` class, you can choose the package you intended to use.

Next, within the `MyFirstAndroidApp` class, declare a constant string that you use to tag all logging messages from this class. You can use the LogCat utility within Eclipse to filter your logging messages to this `DEBUG_TAG` tag string:

```
private static final String DEBUG_TAG= "MyFirstAppLogging";
```

Now, within the `onCreate()` method, you can log something informational:

```
Log.i(DEBUG_TAG,
    "In the onCreate() method of the MyFirstAndroidAppActivity Class");
```

While you're here, you must comment out your previous `forceError()` call so that your application doesn't fail. Now you're ready to run `MyFirstAndroidApp`. Save your work and debug it in the emulator. You notice that your logging messages appear in the LogCat listing, with the Tag field MyFirstAppLogging (see Figure 3.19).

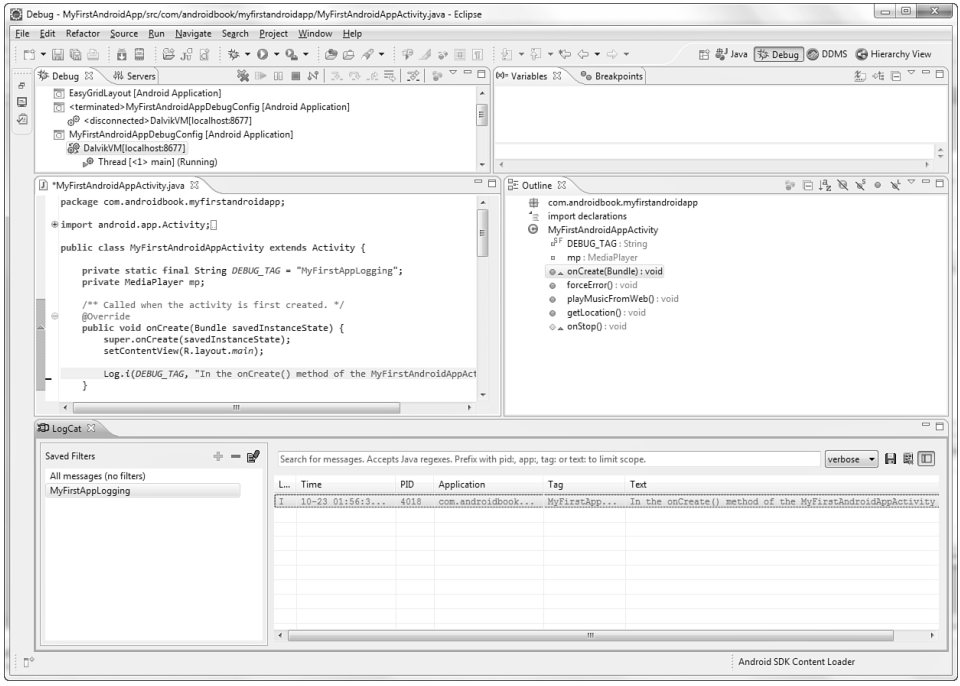

Figure 3.19 A LogCat log for My First Android App.

Adding Some Media Support to Your Application

Next, let's add some pizzazz to `MyFirstAndroidApp` by having the application play an MP3 music file. Android media player features are found in the `MediaPlayer` class of the `android.media` package.

You can create `MediaPlayer` objects from existing application resources or by specifying a target file using a Uniform Resource Identifier (URI). For simplicity, we begin by accessing an MP3 using the `Uri` class from the `android.net` package.

Some methods in the `android.media.MediaPlayer` and `android.net.Uri` classes are shown in Table 3.2.

Table 3.2 **Commonly Used MediaPlayer and URI Parsing Methods**

Method	Purpose
`MediaPlayer.create()`	Creates a new media player with a given target to play
`MediaPlayer.start()`	Starts media playback
`MediaPlayer.stop()`	Stops media playback
`MediaPlayer.release()`	Releases the media player resources
`Uri.parse()`	Instantiates a `Uri` object from an appropriately formatted URI address

To add MP3 playback support to `MyFirstAndroidApp`, edit the file `MyFirstAndroidApp.java`. First, you must add the appropriate import statements for the `MediaPlayer` class:

```
import android.media.MediaPlayer;
import android.net.Uri;
```

Next, within the `MyFirstAndroidApp` class, declare a member variable for your `MediaPlayer` object:

```
private MediaPlayer mp;
```

Now, create a new method called `playMusicFromWeb()` in your class and make a call to this method in your `onCreate()` method. The `playMusicFromWeb()` method creates a valid `Uri` object, creates a `MediaPlayer` object, and starts the MP3 playing. If the operation should fail for some reason, the method logs a custom error with your logging tag. The `playMusicFromWeb()` method should look something like this:

```
public void playMusicFromWeb() {
    try {
        Uri file = Uri.parse("http://www.perlgurl.org/podcast/archives"
            + "/podcasts/PerlgurlPromo.mp3");
        mp = MediaPlayer.create(this, file);
        mp.start();
    }
    catch (Exception e) {
        Log.e(DEBUG_TAG, "Player failed", e);
    }
}
```

As of Android 4.0 (API Level 14), using the `MediaPlayer` class to access media content on the Web requires the `INTERNET` permission to be registered in the application's Android manifest file. Finally, your application requires special permissions to access location-based functionality. You must register this permission in your `AndroidManifest.xml` file. To add permissions to your application, perform the following steps:

1. Double-click the AndroidManifest.xml file.
2. Switch to the Permissions tab.
3. Click the Add button and choose Uses Permission.
4. In the right pane, select `android.permission.INTERNET`.
5. Save the file.

Later on, you'll learn all about the various `Activity` states and callbacks that could contain portions of the `playMusicFromWeb()` method. For now, know that the `onCreate()` method is called every time the user navigates to the `Activity` (forward or backward) and whenever he or she rotates the screen or causes other device configuration changes. This doesn't cover all cases, but will work well enough for this example.

And finally, you want to cleanly exit when the application shuts down. To do this, you need to override the `onStop()` method of your `Activity` class and stop the `MediaPlayer` object and release its resources. The `onStop()` method should look something like this:

```
protected void onStop() {
    if (mp != null) {
        mp.stop();
        mp.release();
    }
    super.onStop();

}
```

Tip

In Eclipse, you can right-click within the class and choose Source (or press Alt+Shift+S). Choose the option Override/Implement Methods and select the `onStop()` method.

Now, if you run `MyFirstAndroidApp` in the emulator (and you have an Internet connection to grab the data found at the URI location), your application plays the MP3. When you shut down the application, the `MediaPlayer` is stopped and released appropriately.

Adding Location-Based Services to Your Application

Your application knows how to say "Hello" and play some music, but it doesn't know where it's located. Now is a good time to become familiar with some simple location-based calls to get the GPS coordinates. To have some fun with location-based services and maps integration, you will use some of the Google applications available on typical Android devices—specifically, the Maps application. You do not need to create another AVD, because you included the Google APIs as part of the target for the AVD you already created.

Configuring the Location of the Emulator

The emulator does not have location sensors, so the first thing you need to do is seed your emulator with some GPS coordinates. You can find the exact steps for how to do this in Appendix A, in the section "Configuring the GPS Location of the Emulator." After you have configured the location of your emulator, the Maps application should now display your simulated location, as shown in Figure 3.20.

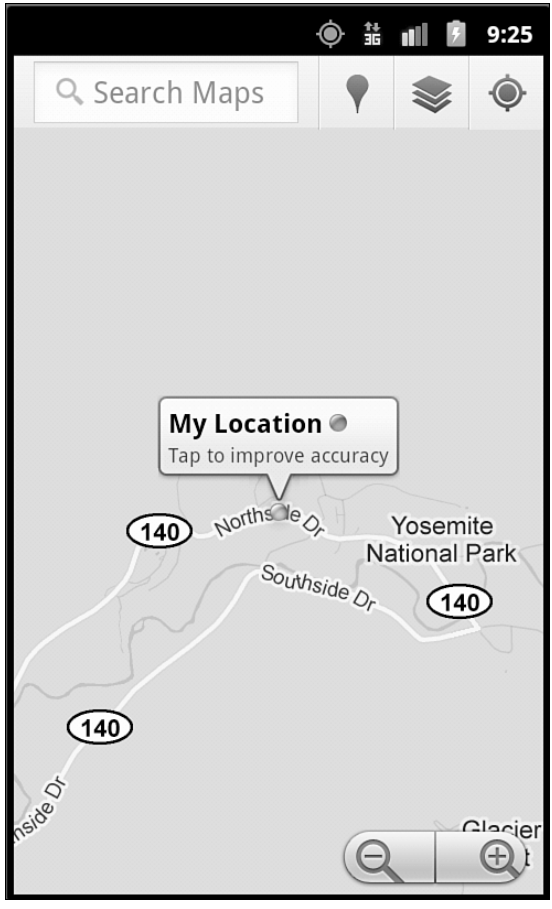

Figure 3.20 Setting the location of the emulator to Yosemite Valley.

Your emulator now has a simulated location: Yosemite Valley!

Finding the Last Known Location

To add location support to `MyFirstAndroidApp`, edit the file `MyFirstAndroidApp.java`. First, you must add the appropriate import statements:

```
import android.location.Location;
import android.location.LocationManager;
```

Now, create a new method called `getLocation()` in your class and make a call to this method in your `onCreate()` method. The `getLocation()` method gets the last known location on the device and logs it as an informational message. If the operation fails for some reason, the method logs an error.

The `getLocation()` method should look something like this:

```
public void getLocation() {
    try {
        LocationManager locMgr = (LocationManager)
            getSystemService(LOCATION_SERVICE);
        Location recentLoc = locMgr.
            getLastKnownLocation(LocationManager.GPS_PROVIDER);
        Log.i(DEBUG_TAG, "loc: " + recentLoc.toString());
    }
    catch (Exception e) {
        Log.e(DEBUG_TAG, "Location failed", e);
    }
}
```

Finally, your application requires special permissions to access location-based functionality. You must register this permission in your `AndroidManifest.xml` file. To add location-based service permissions to your application, perform the following steps:

1. Double-click the `AndroidManifest.xml` file.
2. Switch to the Permissions tab.
3. Click the Add button and choose Uses Permission.
4. In the right pane, select `android.permission.ACCESS_FINE_LOCATION`.
5. Save the file.

Now, if you run My First Android App in the emulator, your application logs the GPS coordinates you provided to the emulator as an informational message, viewable in the LogCat pane of Eclipse.

Debugging Your Application on the Hardware

You mastered running applications in the emulator. Now let's put the application on real hardware. First, you must register your application as debuggable in your `AndroidManifest.xml` file. To do this, perform the following steps:

1. Double-click the `AndroidManifest.xml` file.
2. Change to the Application tab.
3. Set the Debuggable application attribute to true.
4. Save the file.

You can also modify the `application` element of the `AndroidManifest.xml` file directly with the `android:debuggable` attribute, as shown here:

```
<application ... android:debuggable="true">
```

Now, connect an Android device to your computer via USB and re-launch the Run configuration or Debug configuration of the application. Because you chose Manual mode for the configuration, you should now see a real Android device listed as an option in the Android Device Chooser (see Figure 3.21).

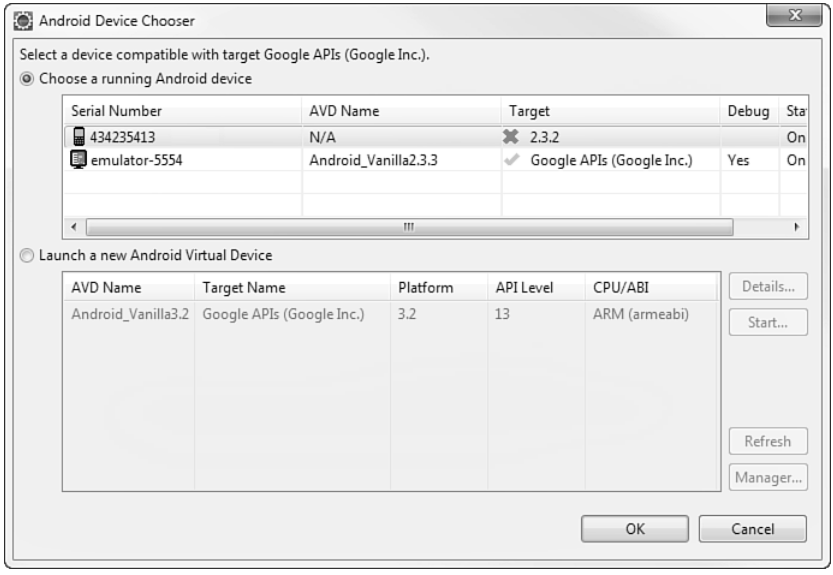

Figure 3.21 Android Device Chooser with USB-connected Android device.

Choose the Android device as your target, and you see that the My First Android App application gets loaded onto the Android device and launched, just as before. Provided you have enabled the development debugging options on the device, you can debug the application here as well. You can tell the device is actively using a USB debugging connection, because a little Android bug-like icon appears in the notification bar ( ). Figure 3.22 shows a screenshot of the application running on a real device (in this case, a smartphone running Android 2.3.2).

Debugging on the device is much the same as debugging on the emulator, but with a couple of exceptions. You cannot use the emulator controls to do things such as send an SMS or configure the location to the device, but you can perform real actions (true SMS, actual location data) instead.

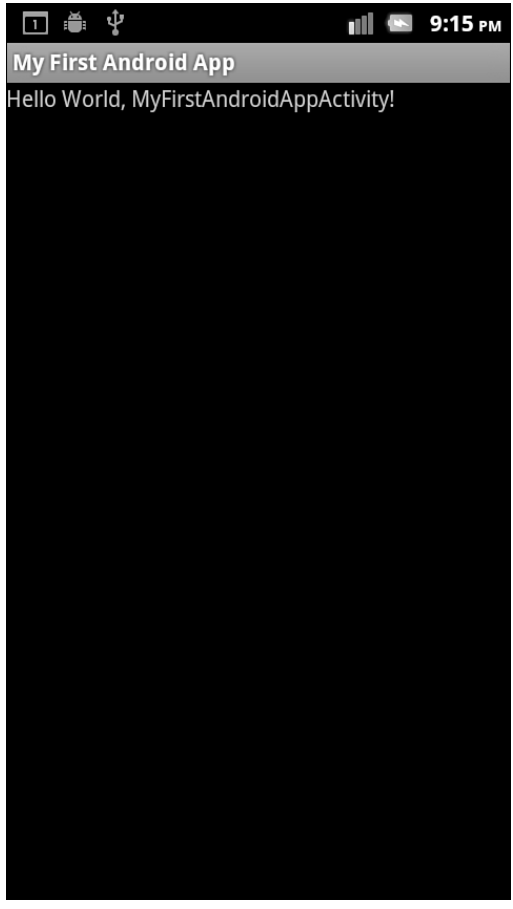

Figure 3.22 My First Android App running on Android device hardware.

Summary

This chapter showed you how to add, build, run, and debug Android projects using Eclipse. You started by testing your development environment using a sample application from the Android SDK and then you created a new Android application from scratch using Eclipse. You also learned how to make some quick modifications to the application, demonstrating some exciting Android features you learn about in future chapters.

In the next few chapters, you learn about the tools available for use developing Android applications and then focus on the finer points about defining your Android

application using the application manifest file and how the application lifecycle works. You also learn how to organize your application resources, such as images and strings, for use within your application.

References and More Information

Android SDK Reference regarding the application `Activity` class:
http://d.android.com/reference/android/app/Activity.html
Android SDK Reference regarding the application `Log` class:
http://d.android.com/reference/android/util/Log.html
Android SDK Reference regarding the application `MediaPlayer` class:
http://d.android.com/reference/android/media/MediaPlayer.html
Android SDK Reference regarding the application `Uri` class:
http://d.android.com/reference/android/net/Uri.html
Android SDK Reference regarding the application `LocationManager` class:
http://d.android.com/reference/android/location/LocationManager.html
Android Dev Guide: "Developing on a Device":
http://d.android.com/guide/developing/device.html
Android Resources: "Common Tasks and How to Do Them in Android":
http://d.android.com/resources/faq/commontasks.html
Android Sample Code: "Snake":
http://d.android.com/resources/samples/Snake

This page intentionally left blank

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X–Z