

BONUS
KitKat Chapter
Available Online



LEARNING **Android™** Application PROGRAMMING

A Hands-On Guide to Building Android Applications



JAMES TALBOT
JUSTIN McLEAN

FREE SAMPLE CHAPTER



SHARE WITH OTHERS

Praise for *Learning Android™ Application Programming*

“James Talbot and Justin McLean have done excellent work creating this beginner’s tutorial. The hands-on focus of the book takes the reader from installing the development environment to writing the app and finally publishing the app. Topics include what most developers would want to know, from basic app structure and function, styling the app, testing and optimization, to using social media.”

—*Matthew Boles, learning specialist, Brightcove Inc.*

“*Learning Android Application Programming* is a treasure trove of holistic information on developing applications for Android devices. It should satisfy those who prefer detailed descriptions as well as those who enjoy a book chock full of high-quality code samples. This book is a one-stop shop to take the reader from zero to app publish via a most relevant sample application. Throughout the book, the sample app is built, refactored, and optimized as the reader picks up all the necessary concepts and skills needed to become a true Android developer. Outstanding work, Justin and James!”

—*Jun Heider, senior architect and development manager, RealEyes Media*

“A unique book that iteratively covers every aspect—requirements, design, developing, testing, and publishing—of a production-grade Android application.”

—*Romin Irani, owner, Mind Storm Software*

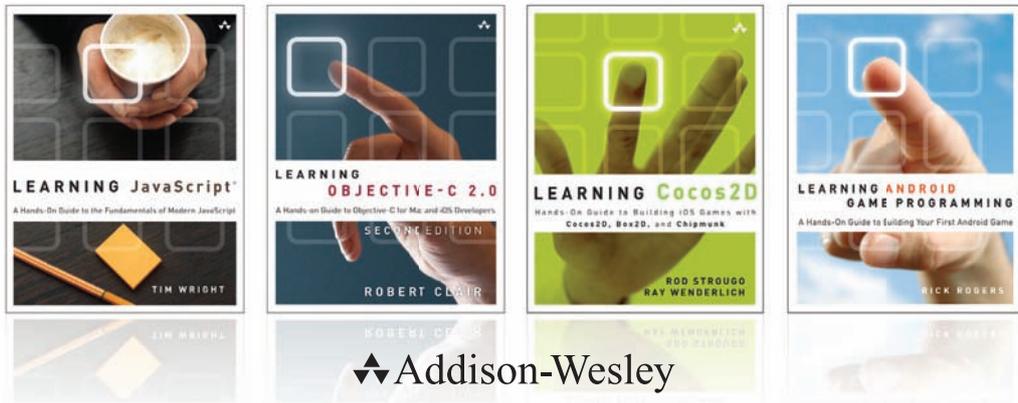
“*Learning Android Application Programming* covers a rich variety of commonly encountered scenarios when approaching the Android development platform. Newcomers can step through the provided examples in an easily approachable format, while those who are more familiar with Android will find many useful nuggets scattered throughout. Everything is written in an understandable way and demonstrated through concrete examples, which can be immediately applied to a multitude of projects—great stuff!”

—*Joseph Labrecque, senior interactive software engineer, University of Denver*

This page intentionally left blank

Learning Android™ Application Programming

Addison-Wesley Learning Series



Visit informit.com/learningseries for a complete list of available publications.

The **Addison-Wesley Learning Series** is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

 Addison-Wesley

 **informIT**
the trusted technology learning source

 **Safari**
Books Online

Learning Android™ Application Programming

A Hands-On Guide to Building
Android Applications

James Talbot

Justin McLean

◆ Addison-Wesley

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

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Talbot, James.

Learning Android application programming : a hands-on guide to building Android applications / James Talbot, Justin McLean.

pages cm

Includes index.

ISBN 978-0-321-90293-1 (pbk. : alk. paper)—ISBN 0-321-90293-9 (pbk. : alk. paper)

1. Android (Electronic resource) 2. Application software—Development. 3. Mobile computing.

I. McLean, Justin. II. Title.

QA76.76.A65T35 2014

004.16—dc23

2013037213

Copyright © 2014 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

Code Listings:

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

The URL <http://www.apache.org/licenses/LICENSE-2.0> has the full terms and conditions of the license.

ISBN-13: 978-0-321-90293-1

ISBN-10: 0-321-90293-9

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

First printing, December 2013

Copyright page continues on page 395.

Editor-in-Chief

Mark L. Taub

Executive Editor

Laura Lewin

Development Editor

Michael Thurston

Managing Editor

John Fuller

Full-Service

Production Manager

Julie B. Nahil

Copy Editor

Betsy Hardinger

Indexer

John S. Lewis

Proofreader

Rebecca Rider

Technical Reviewers

Romin Irani

Douglas Jones

Prashant Thakkar

Editorial Assistant

Olivia Basegio

Cover Designer

Chuti Prasertsith

Compositor

Kim Arney



*I'd like to thank my family and friends,
as well as my colleagues at Adobe Systems.
This book is dedicated to my brand new niece, Lenora Talbot,
who is entering a world that is forever changed
by the mobile revolution.*

—James Talbot

*I'd like to thank my family, friends,
and all the new people I've met over the last year while traveling,
speaking at conferences, and writing this book.*

Life would be a boring place without you.

*Parts of this book were written in New York City; Los Angeles;
San Francisco; Portland; Gloucester, MA; Denver; St. Louis; Sydney;
Hobart; Perth; Melbourne; Brisbane; up Bostock Road (near Tucabia);
Cologne; Berlin; outside Arklow (near Dublin); London;
Ammanford in Wales; and Edinburgh.*

*It's been a fun adventure, and I hope you enjoy the book
as much as I've enjoyed working on it.*

—Justin McLean



This page intentionally left blank

Contents at a Glance

Preface	xix
Acknowledgments	xxi
About the Authors	xxiii
1 An Introduction to Android Development	1
2 Kicking the Tires: Setting Up Your Development Environment	13
3 Putting On the Training Wheels: Creating Your First Android Application	29
4 Going for Your First Ride: Creating an Android User Interface	51
5 Customizing Your Bike: Improving Android Application Usability	81
6 Pimping Your Bike: Styling an Android Application	125
7 Are We There Yet? Making Your Application Location Aware	165
8 Inviting Friends for a Ride: Social Network Integration	223
9 Tuning Your Bike: Optimizing Performance, Memory, and Power	249
10 Taking Off the Training Wheels: Testing Your Application	285
11 Touring France: Optimizing for Various Devices and Countries	327
12 Selling Your Bike: Using Google Play and the Amazon Appstore	363
Index	377

This page intentionally left blank

Contents

Preface	xix
Acknowledgments	xxi
About the Authors	xxiii
1 An Introduction to Android Development	1
Understanding the Android Difference	2
Building Native Applications	2
Understanding the History of Android	3
Using the Android User Interface	8
Understanding Android Applications	10
Introducing Google Play	10
Summary	12
2 Kicking the Tires: Setting Up Your Development Environment	13
Installing the Java JDK and JRE on Windows	14
Understanding Java Versions	16
Installing the Eclipse IDE on Windows	16
Installing Eclipse	16
Configuring the Java JRE in Eclipse	17
Getting Familiar with Eclipse	17
Installing the Android SDK on Windows	19
Installing the Android Developer Tools Plug-in on Windows	21
Installing and Using Java on a Mac	24
Downloading and Installing the JDK on a Mac	24
Downloading and Installing the Eclipse IDE on a Mac	25
Downloading and Installing the Android SDK on a Mac	25
Installing the Android Developer Tools Plug-in on a Mac	26
Summary	27

3 Putting On the Training Wheels: Creating Your First Android Application 29

Creating an Android Application	29
Running Your Android Project	32
Creating an Android Virtual Device	32
Running an Application on the AVD	33
Best Practices for Using an Android Virtual Device	34
Installing an Android Application on an Actual Device	36
Working with Lint in an Android Project	37
Understanding the Android Project Files	37
Understanding the Layout XML Files	38
Understanding the Resource XML File	39
Using IDs in XML Files and Their Effect on Generated Files	40
Understanding the Activity File	42
Understanding the Activity Lifecycle	44
Getting Access to the TextView Within the Activity	45
Using Logging in Your Application	46
Understanding the Android Manifest File	49
Summary	50

4 Going for Your First Ride: Creating an Android User Interface 51

Refactoring Your Code	51
Implementing Strict Mode	54
Creating a Simple User Interface	55
Using Linear Layouts	56
Creating Button Event Handlers	60
Updating the Timer Display	63
Displaying a Running Timer	65
Understanding the Activity Lifecycle	68
Exploring the Android Activity Lifecycle	70
Fixing Activity Lifecycle Issues	72
Making an Android Device Vibrate	72

Saving User Preferences	74
Creating a New Activity	75
Showing a New Activity	75
Saving an Application's State	76
Using Shared Preferences	79
Summary	80

5 Customizing Your Bike: Improving Android

Application Usability	81
Refactoring Your Code	82
Improving the Setting Activity	88
Showing Toast Pop-Ups	88
Returning from the Settings Activity with a Back Button	92
Action Bars and Menus	94
Creating a Menu	95
Creating an Action Bar	97
Going Home	99
Using Notifications	101
Creating a Notification	101
Showing or Replacing a New Notification	104
Showing Notifications at Regular Intervals	104
Creating a Database	107
Creating a Data Model	108
Creating a Database and Its Tables	109
Checking Table Creation	112
Creating Relationships Between Tables	113
Creating a Routes ListView	118
Summary	122

6 Pimping Your Bike: Styling an Android

Application	125
Refactoring Your Application	126
Understanding Screen Differences	126
Understanding Screen Sizes and Densities	127
Knowing the Devices Out There	128
Making Your Application Resolution Independent	129

Using Configuration Qualifiers	132
Creating Launcher Icons	134
Creating Notification Icons	136
Making Apps Look Good on Different Screen Sizes	137
Using Resource Dimensions	140
Changing Text Size in Java	142
Changing the Layout for Landscape Mode	144
Changing the Layout for Tablets	145
Creating a Side-by-Side View	146
Using Styles and Themes	149
Enabling Night Mode	151
Changing Themes	153
Detecting Light Levels	158
Dealing with Erratic Sensor Values	160
Summary	162

7 Are We There Yet? Making Your Application

Location Aware	165
Refactoring Your Code	165
Finding the Device's Location	169
Testing GPS in a Virtual Device	175
How Accurate Is the GPS Location?	176
Improving the User Experience When Using GPS Location	178
Displaying Google Maps	181
Dealing with Inaccurate Location Data	190
Storing GPS Data	196
Inserting, Updating, and Deleting Data	197
Updating the Model	200
Using the Database in Your Application	205
Displaying GPS Data	209
Working with List Activities	209
Displaying GPS Data in Google Maps	214
Summary	220

8	Inviting Friends for a Ride: Social Network Integration	223
	Refactoring Your Code	223
	Integrating Photos into an Android Application	224
	Taking a Photograph	224
	Checking Whether You Can Take a Photograph	226
	Displaying a Photograph in Your Application	231
	Getting Results from Activities	237
	Sharing Content with Friends	242
	Displaying a Chooser	242
	Sharing Text and Photos	245
	Summary	248
9	Tuning Your Bike: Optimizing Performance, Memory, and Power	249
	Refactoring Your Code	249
	Running Your Application as a Service	250
	Handling Orientation Changes	251
	Creating a Service	254
	Improving Battery Life	267
	Determining Power Usage	268
	Reacting to Power Levels	270
	Checking the Battery Regularly	276
	Speeding Up Databases	278
	Speeding Up Databases with Indexes	278
	Speeding Up Databases with Asynchronous Tasks	280
	Summary	284
10	Taking Off the Training Wheels: Testing Your Application	285
	Refactoring Your Code	285
	Testing with JUnit	286
	Creating a New Test Application	286
	Increasing Test Coverage	292

Speeding Up Your Tests	294
Making Testing Easier by Refactoring	297
Testing with Android JUnit Extensions	299
Testing Android Activities	300
Creating a Mock Application	302
Testing an Activity Lifecycle	305
Testing an Activity Further	308
Testing by Interacting with the UI	309
Testing Services	310
Using Monkey Testing	313
Running Tests Automatically	316
Running Tests from the Command Line	316
Installing Jenkins	318
Using Version Control with Git	319
Overview of Git Bash Commands	321
Using Jenkins	322
Testing on a Wide Range of Devices	323
Summary	325
11 Touring France: Optimizing for Various Devices and Countries	327
Refactoring Your Code	327
Going International	329
Supporting Various Languages	330
Starting with a Rough Machine Translation	331
Improving the Translation with Help from Users	335
Adding More Languages	337
Accommodating Various Dialects	342
Adding Language Region Codes	342
Dealing with Word Variations: Route, Path, Trail, and Track	343
Handling Various Language Formats	344
Supporting Right-to-Left Layouts	344
Dealing with Variations in Dates, Numbers, and Currencies	346
Enabling Backward Compatibility	348

Using the Android Support Library	348
Android Version Checking	349
Building for Various Screen Sizes	352
Using Fragments	355
Summary	361

12 Selling Your Bike: Using Google Play and the Amazon Appstore 363

Building Your Media Strategy	363
Using Google Play	364
Implementing Google Licensing Using Services and APIs	368
Employing Advertising in Your Application	369
Using the Amazon Appstore	373
Summary	376

Index	377
--------------	------------

This page intentionally left blank

Preface

This is a book about learning how to program an Android application from start to finish. It assumes that you have some web development or programming experience but may not be familiar with the Java language or the Android operating system or have working knowledge of the Android API/SDK. This book teaches you best practices for programming Android applications and explains how to solve real-world issues such as device fragmentation. You'll learn how to code your application to work on the widest range of Android OSs while still taking advantage of the latest Android features, and you'll explore how to use (often inaccurate) data from sensors. You'll discover how to preserve the battery life of your device and how to make your application easily work in multiple countries and languages.

Each chapter builds upon the preceding chapter, step by step, until you have a complete working application. This book is best read in order, but you can skip around if you already understand the content in a chapter, because the code for each chapter can be found on the book's website and on GitHub. However, remember that the goal of this book is to learn by doing, and, if you follow each chapter, you will learn some useful best practices.

This book is aimed at web developers or programmers who may have little or no Android or Java experience and want to know how to write an Android application from start to finish. This book is not an API reference, and it isn't filled with small snippets of unconnected code. Instead, it's a hands-on, learn-as-you-go tutorial that helps you avoid the common traps and pitfalls that new Android developers get themselves into. As you go through each chapter, you'll build the *On Your Bike* Android application, a handy tool for bicycle riders. When you've finished the book, you'll have a complete application, and you will have learned enough to create your own application and publish it in Google Play and the Amazon Appstore.

While working through this book, it's recommended that you have access to an Android device. Although it's possible to work through most of the book using only a computer and the Android emulator, there are some things that will work only on a real device.

The color code in the printed book is meant to be representative of what you will see when you are programming in Eclipse. Colors do not match exactly but are close approximations of what you will see in the Eclipse Development Environment.

Code Examples

The code listings for each chapter can be found at the book's website:

<http://www.androiddevbook.com/code.html>

They are also available on GitHub:

<https://github.com/androiddevbook/onyourbike>

The application can also be found in Google Play:

<https://play.google.com/store/apps/details?id=com.androiddevbook.onyourbike.book>

If you have any questions about the book or the code, please contact the authors at james@androiddevbook.com or justin@androiddevbook.com. You can follow the book on Twitter at [@androiddevbook](https://twitter.com/androiddevbook). The code and more information are on <http://www.androiddevbook.com>.

Acknowledgments

The authors would like to thank the following:

Jorge Hernández, who proposed the initial idea of a cycle computer application and helped us write and format the initial chapters.

Romin Irani, Douglas Jones, and Prashant Thakkar for being the technical reviewers of the book. They picked up many issues, both big and small, during the review process, and the book would not be the quality it is without the time and effort they put into reviewing both the content and the code.

Betsy Hardinger, an absolutely amazing editor who caught many things we did not even think of. We appreciate her professionalism and diligence.

Michael Thurston, the development editor, who suffered through our bad grammar and spelling, inconsistent formatting, and confusing language and structure, and managed to get the draft document into a state fit for publishing.

Olivia Basegio, an incredible asset, who always stepped in and made us feel comfortable with the daunting process of publishing a book.

Laura Lewin, the editor of the project, who kept us on schedule and was tireless in her research and assistance. We owe a lot to her and really appreciate her professionalism.

We would also like to thank Julian Ledger for designing the Android *On Your Bike* icon and for general design guidance in the production of this book.

Finally, for the internationalization chapter we'd like to thank several people who provided translations of the application resource strings into their own language: Kai König (German), Carlos Rovira (Spanish), Frédéric Thomas (French), Christophe Herreman (Dutch), and John Koch (Japanese).

This page intentionally left blank

About the Authors

James Talbot has been with Adobe for more than a decade, on the sales engineering, professional services, and training teams, and has many years of experience in working with object-oriented programming and web applications. He is currently working on constructing exciting web, mobile web, and native Android applications built on top of a Java Content Repository (JCR) based on open source standards. He cowrote *Object-Oriented Programming with ActionScript 2.0* (New Riders Press, 2004) and *Adobe Flex 2: Training from the Source* (Adobe Press, 2006), as well as *Adobe Flex 3: Training from the Source* (Adobe Press, 2008). He has also recorded training videos for Lynda.com and Total Training and has spent extensive time teaching in the classroom. He has deep knowledge of all Adobe web products and has spoken at numerous conferences.

Justin Mclean has been writing code since the early days of the web. For 15 years he has managed his own consulting company, Class Software, and during that time he has worked on hundreds of browser, desktop, and mobile applications. He has seen significant changes of technology in the industry, surviving the browser wars and the dot-com bubble. He is an Apache Flex committer, board member, and release manager and an Adobe Community Professional. He teaches training courses and has spoken at numerous conferences all over the world. In his spare time he tinkers about with open source electronics.

This page intentionally left blank

We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write us directly to let us know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or email address.

Email: laura.lewin@pearson.com

Mail: Reader Feedback

Addison-Wesley Learning Series

800 East 96th Street

Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Going for Your First Ride: Creating an Android User Interface

*Life is like a riding a bicycle,
you don't fall off unless you stop pedaling.*

—Claude Pepper

Now it's time to begin coding the **On Your Bike** application. This Android app will act as a bicycle computer—a device, usually clipped on the handlebars, that helps you keep track of the length and time of your ride. By creating this application, you will learn more about how to code with the Android activity lifecycle, how to code a simple user interface, and how to specify user preferences.

Refactoring Your Code

Because of project time pressures, you often need to make quick changes to code. Over time, these little changes add up, and, as a result, you need to revisit the code before the project is complete. This is known as technical debt. The code base becomes fragile, and it's easy to introduce bugs and more difficult to maintain the code. It's important to have a spring cleaning every now and then to fix the most obvious issues.

It makes sense to rearrange the code at a time when you're not trying to change its functionality, a process referred to as refactoring. Of course, it's also much easier to change functionality when you have clean, refactored code.

When you're undertaking a major refactoring, don't forget to back up your code first, or, better still, keep your code under version control. But don't despair if you get lost and make a mistake with your code: You can always download the code for the

chapter from the **On Your Bike** website (<http://www.androiddevbook.com>) or from GitHub (<https://github.com/androiddevbook/onyourbike>).

The simplest form of refactoring is to rename packages, classes, methods, and variables. You might do this for several reasons.

- Renaming a class, method, or variable will increase the readability or understanding of the existing code.
- Naming wasn't consistent across the application.
- A method's functionality has changed, and it now does something a little different from what its original name indicated. It makes sense to rename the method to something more descriptive.
- You can move duplicate blocks of code into a single new method. This can help implement the Don't Repeat Yourself (DRY) principle, whose primary purpose is to prevent the repetition of information.
- You can break larger methods into several smaller methods so that they can be reused. This will also make the code more readable.

Always remember, your code should be human readable first, and machine readable second. If you've ever had to work on other people's code or returned to code you wrote months ago, you'll be thankful for that readability. If you don't follow this principle, it can result in substantial frustration. You may end up cursing yourself—or the original developer.

Now let's refactor your ongoing project to better describe it. Follow these steps.

1. In the **Package Explorer** view, do the following.
 - Expand the `/src` directory.
 - Right-click the `com.androiddevbook.onyourbike.chapter3` package.
 - Select **Refactor > Rename**.
 - Change the end of the package name from `chapter3` to `chapter4`, as shown in Figure 4.1. Keep the **Update references** checkbox checked.
 - Click **Preview** to check the changes that will take place. You will see that the import statements in **MainActivity** will change and that the package will be renamed.
 - Click **OK** to apply the changes. Ignore any compiler errors that are shown.
2. Perform the same procedure (by right-clicking the filename and selecting **Refactor > Rename**) with the **MainActivity** class, and rename it **TimerActivity**.
3. Locate the `\res\layout\activity_main.xml` file, and rename it **activity_timer.xml**.

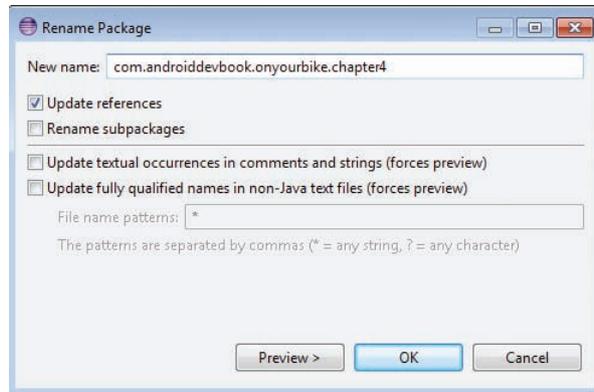


Figure 4.1 Rename Package dialog box in Eclipse

4. Change the call to the `setContentView` method in **TimerActivity** to pass the new activity identifier:

```
setContentView(R.layout.activity_timer);
```

5. After you save the **TimerActivity.java** file, the compilation error will be resolved.
6. Open `\res\values\strings.xml`, and change the following lines to reflect a new application name and a new title.

- Change the value of the string node with an attribute `app_name` to the following:

```
<string name="app_name">On Your Bike - Chapter 4</string>
```

- Change the name attribute `title_activity_main` to `title_activity_timer`, and the node value to the following:

```
<string name="title_activity_timer">Timer</string>
```

7. Double-click on the error in the *Problem* view to open the `AndroidManifest.xml` file. Change the following.

- Change the package name to match the new package:

```
package="com.androiddevbook.onyourbike.chapter4"
```

- Change the activity name to match the new activity class:

```
android:name=".TimerActivity"
```

- Change the activity label to match the new string resource:

```
android:label="@string/title_activity_timer"
```

- From the **Refactor** menu, select **Rename**, and rename the `className` constant in **TimerActivity**. It is better practice to define a variable treated as a constant with uppercase letters and make it private so that it is not visible outside the class:

```
private static String CLASS_NAME;
```

Eclipse will automatically rename all references to the constant.

- Rename the project **On Your Bike Chapter 4** by right-clicking on the project name, selecting **Refactor -> Rename**, entering the new name, and clicking **OK**. It is a good idea to clean your project after making all the changes to make sure that everything has been recompiled and to double-check that there are no errors. You do this by selecting **Project > Clean**.

Implementing Strict Mode

When you're first programming for Android, you need to be aware of several gotchas that may trip you up. For example, it's common to accidentally block the user interface thread and cause your application to perform badly or, even worse, to become unresponsive. Strict mode was added to the Android SDK to identify issues like this. It's a good idea, especially when you're starting out, to always turn on Strict mode.

Strict mode is flexible in that you can filter issues so that it reports only the ones you're interested in and, when those issues occur, what sort of action should be taken.

You can take the following actions:

- Logging the issue to LogCat
- Flashing the device's screen
- Stopping the application
- Opening a dialog box

Setting up Strict mode in your application is straightforward.

- To enable Strict mode, add the code in Listing 4.1 after the call to `Log.d` in the `onCreate` method of `TimerActivity`.

Listing 4.1 Turning On Strict Mode in `onCreate`

```
if (BuildConfig.DEBUG) {
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
        .detectAll().penaltyLog().build());
    StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
        .detectAll().penaltyLog().penaltyDeath().build());
}
```

This code will detect all issues with threading and display them to the *LogCat* view. It will also detect common memory leaks, log them, and stop the application. Note that the `Builder` constructor and all the various detect and penalty methods return the current instance of `builder`. This is known as function chaining. In this way, methods can be called together one after another to make the code more readable and concise.

2. A few errors will show in the *Problem* view. Run **Quick fix** `StrictMode` to add the import statement:

```
import android.os.StrictMode;
```

Creating a Simple User Interface

At this point, your activity `_timer` activity is using as its base tag the `RelativeLayout` view group. By using the `RelativeLayout` class, you're telling the app to position the views in relation to how other views are positioned. For example, the position of views could be determined by whether the views are to the right or left of another view, below or above another view, centered in the view group, aligned to the left or right of each other, or even aligned to the bottom or top of the view group.

The values for the layout properties are either a Boolean or an ID that references another view. In the XML layout, they can be declared in any order. For example, if `android:layout_centerVertical` is set to `true`, then the top edge of the view will match the top edge of the parent. If `android:layout_below` is set, then the top edge of the view will be below the view specified with a resource ID—for example, `android:layout_below="@id/name"`. If `android:layout_toRightOf` is set, then the left edge of the view will be to the right of the view with the resource ID.

Once you have indicated the position of the views in a view group, you can then specify the layout width and layout height. These measurements can be an exact number and a unit.

Possible units of measurement include the following.

- Density-independent pixels (dp): Use to make UI elements the same size on different screen densities.
- Pixels (px): Try to use dp instead.
- Scale-independent pixels (sp): Use for font sizes that scale according to the user preference and the screen density.
- Points (pt): Try to use sp instead.
- Millimeters (mm) and inches (in): Avoid if possible.

You can also specify the height and width in terms of the view's actual size or the view group's size; to do this, set the width or height layout attribute to

`wrap_content` or `match_parent`. This gives you even more flexibility in designing layouts for devices of various sizes. (Note that `match_parent` was called `fill_parent` in earlier versions of the SDK, so you may come across this in old code.) The `wrap_content` attribute makes the view as big as it needs to be, so the view group layout may include gaps; `match_parent` also makes the view resize, so there are no gaps in the view group's layout except for the padding.

There are other concepts that come into play when you're laying out views. `Weight` describes how the total width or height is shared between multiple views. For example, each child view is given a proportion of its weight over the total weight of all views. If all child views have the same weight, then all of them will have the same height and width. However, if a child view has a weight of 2 and other child views have a weight of 1, then the first child view will be twice as high and twice as wide as the other views.

If any of the child views also has a width or height, then the remaining space is divided by the weights; in the preceding example, the first child would be proportionally wider but not twice as wide as the other views. You often need to experiment with the right combination of width, height, and weight to get something that works for each view.

It makes sense to do one of two things: either (1) express weight in terms of how much bigger or smaller a view is compared to its siblings or (2) make the weights add up to 100 so that the weight can be thought of as a percentage. You should use whatever makes sense in the layout.

Another view group is `LinearLayout`. A `LinearLayout` enables you to position views vertically (one view above the other on the screen) or horizontally (the views side by side). To control whether the views inside a `LinearLayout` are positioned horizontally or vertically, set the `orientation` attribute to `horizontal` or `vertical`.

Note that layouts are defined in this way so that the screen size of an activity is mostly irrelevant and activities scale and resize to fit on a wide range of screen sizes and densities. In this way, your app can display correctly on all the different Android devices out there.

Other layout view groups include `GridView`, `ListView`, and `WebView`. As you might expect, `GridView` displays items in a grid, `ListView` displays views in a vertical list, and `WebView` displays web pages. Laying out a UI is a complex topic, and you will learn much more about it as you begin to build the application.

Using Linear Layouts

The basic display on a bicycle computer includes a timer that tracks how long you've been riding. You will build this functionality in this section. The first step is to build a user interface that will include a Start button as well as a Stop button for the timer. The timer output will appear on the text view you have already created. To build this functionality, follow these steps.

1. Edit the existing `TextView` in the `activity_timer.xml` file. Remove the line that sets the `android:text` attribute. (The text will no longer be hard-coded but instead will be dynamic and changed through code you will add later.)
2. Change the `android:id` to the value `@+id/timer`:

```
<TextView
    android:id="@+id/timer"
```

3. Change the `toolContext` to be the `TimerActivity` class by assigning it a value `.TimerActivity`:

```
tools:context=".TimerActivity"
```

4. Add a `LinearLayout` below the `TextView`:

```
<LinearLayout>
</LinearLayout>
```

5. In the linear layout you just added, you will add two buttons. The buttons need to stretch horizontally. To do this, change the `android:layout_width` to `match_parent`. For the buttons to be as high as they need to be, set the `android:layout_height` to `wrap_content`. Set the `android:orientation` to `horizontal` so that the buttons are side by side:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
```

6. Still inside the `LinearLayout` tag, align the buttons at the bottom of the screen by assigning the `android:layout_alignParentBottom` to `true`:

```
android:layout_alignParentBottom="true"
```

7. Also inside the `LinearLayout` tag, add the `Start` button inside the linear layout, give it an `android:id` of `@+id/start_button`, and set the `android:layout_width` and `android:layout_height` to `wrap_content`:

```
<Button
    android:id="@+id/start_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

8. Still inside the `LinearLayout` tag, add the `Stop` button after the `Start`, give it an `android:id` of `@+id/stop_button`, and set the `android:layout_width` and `android:layout_height` to `wrap_content`:

```
<Button
    android:id="@+id/stop_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

- The two buttons need to be the same size, so set the `android:layout_weight` on both to 1:

```
android:layout_weight="1"
```

- Click on the **Graphical Layout** view to check that there are no errors; that the view consists of a `TextView` in the center of the layout; and that there are two buttons of equal size at the bottom of the layout, as shown in Figure 4.2.

- To the first button, add an `android:text` value of `@string/start_button`:

```
android:text="@string/start_button"
```

- Add the same attribute to the second button with a value of `@string/stop_button`:

```
android:text="@string/stop_button"
```

- Add the two created resources to `values/strings.xml`:

```
<string name="start_button">Start</string>
<string name="stop_button">Stop</string>
```

Your layout code should now look like Listing 4.2.

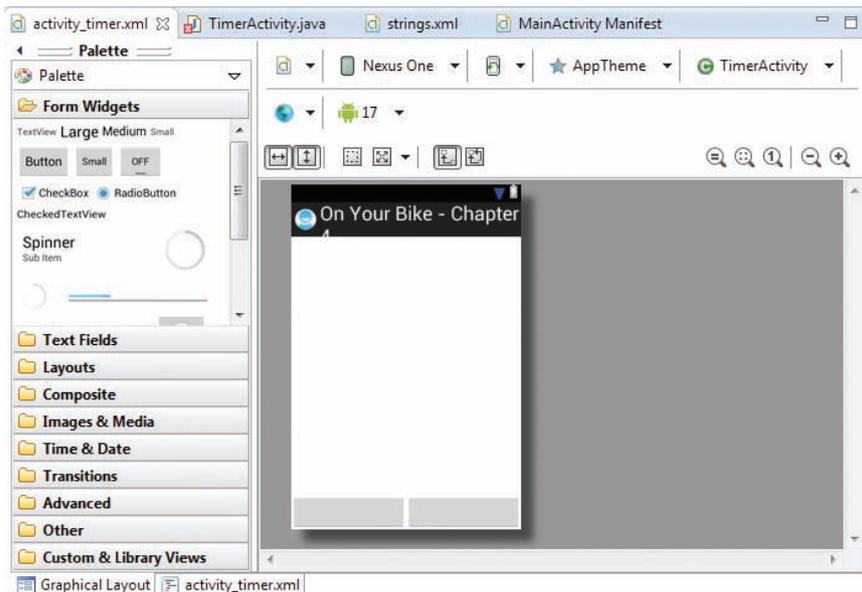


Figure 4.2 Graphical Layout view showing two blank buttons

Listing 4.2 Linear Layout Containing Two Buttons

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/timer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        tools:context=".TimerActivity" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:orientation="horizontal">

        <Button
            android:id="@+id/start_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/start_button" />

        <Button
            android:id="@+id/stop_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/stop_button" />
    </LinearLayout>

</RelativeLayout>
```

14. Open `TimerActivity.java` and either correct or run **Quick fix** to address the error by changing `hello` to `timer`.
15. **Debug** your application. The activity should be displayed, as shown in Figure 4.3. You can click both buttons, but they don't do anything yet.

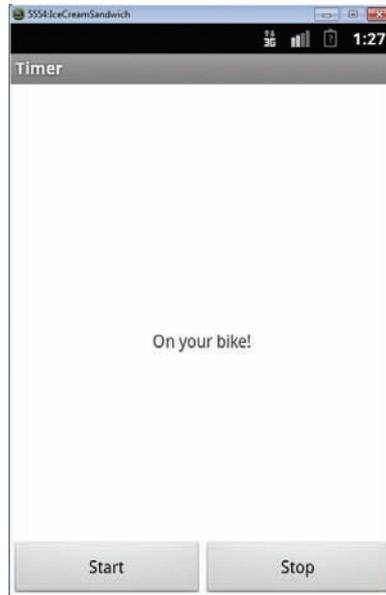


Figure 4.3 Debugging application showing buttons

Creating Button Event Handlers

To make the buttons do something, you need to add event handlers to the buttons that detect when they are clicked, and you need to supply the method to be called. There are several ways of doing this with the Android SDK, but first let's take the simple approach and add the handlers to the layout.

1. Open the activity_timer.xml layout file, and locate the two buttons you added earlier. Add the two click handlers to the appropriate buttons by setting android:onClick to the name of the methods you want called when the buttons are clicked. Call the two methods clickedStart and clickedStop, as shown in Listing 4.3.

Listing 4.3 Adding Click Handlers to Two Buttons

```
<Button
    android:id="@+id/start_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/start_button"
    android:onClick="clickedStart" />
```

```

<Button
    android:id="@+id/stop_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/stop_button"
    android:onClick="clickedStop" />

```

2. Add the `clickedStart` and `clickedStop` methods to the `TimerActivity` class, logging that the methods have been called. Run **Quick fix** to import the `View` class. See Listing 4.4.

Listing 4.4 Adding Click Handlers Methods

```

public void clickedStart(View view) {
    Log.d(CLASS_NAME, "Clicked start button.");
}

public void clickedStop(View view) {
    Log.d(CLASS_NAME, "Clicked stop button.");
}

```

3. **Debug** the application. Click each button to make sure the click log messages are displayed in the *LogCat* view, as shown in Figure 4.4.

Note that if the method names are incorrect (if they don't match what is in the layout XML), then the application will compile and run with no warnings or errors, but you will get a run time exception (RTE) when clicking on the button. This is the downside of specifying handlers this way, but it's easy enough to avoid with a little care and testing.

4. Add the following class properties at the top of the `TimerActivity` class declaration:

```

protected TextView counter;
protected Button start;
protected Button stop;

```

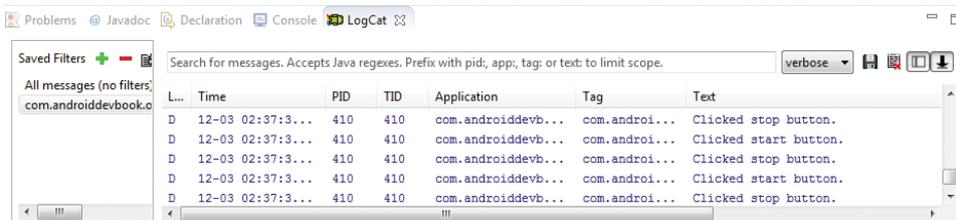


Figure 4.4 LogCat view showing Start and Stop button logs

- Change the `onCreate` method to assign each of these variables to match the corresponding view in the layout. To do this in each case, call `findViewById`, passing the automatically generated identifier for that view. This must be done after the `setContentView` call; otherwise, you get an RTE when the application is run. Also, change the text view `findViewById` to refer to the new timer variable:

```
counter = (TextView) findViewById(R.id.timer);
start = (Button) findViewById(R.id.start_button);
stop = (Button) findViewById(R.id.stop_button);
```

- Remove the `hello.setText` line. The text of this text view will now be set through code.
- Create a new class property called `timerRunning` to store the state of the timer and whether or not it has been started. This in turn determines whether the buttons are enabled or disabled.

```
protected boolean timerRunning;
```

- Add a new method called `enableButtons` to toggle which button (Start or Stop) is enabled depending on the value of `timerRunning`:

```
protected void enableButtons() {
    Log.d(CLASS_NAME, "Set buttons enabled/disabled.");
    start.setEnabled(!timerRunning);
    stop.setEnabled(timerRunning);
}
```

- Call `enableButtons` after the calls to `findViewById` in `onCreate` and in the `clickedStart` and `clickedStop` methods.
- Before the call to `enableButtons`, set the property `timerRunning` to true in `clickedStart`, and to false in `clickedStop`. Your two event handlers should now look like Listing 4.5.

Listing 4.5 Button `onClick` Event Handlers

```
public void clickedStart(View view) {
    Log.d(CLASS_NAME, "Clicked start button.");
    timerRunning = true;
    enableButtons();
}

public void clickedStop(View view) {
    Log.d(CLASS_NAME, "Clicked stop button.");
    timerRunning = false;
    enableButtons();
}
```

- Run** the application. The buttons should now toggle to the one that is enabled when it is clicked, as shown in Figure 4.5.

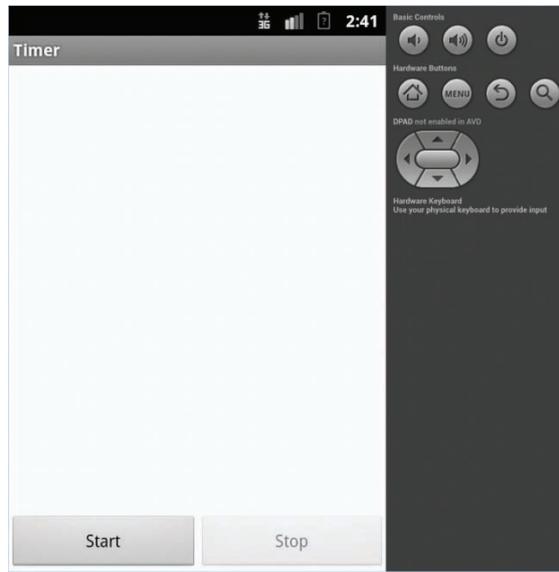


Figure 4.5 Debugging application showing enabled button

Updating the Timer Display

For the application to work as a bicycle computer, the counter needs to be updated frequently. This update is based on how much time has passed since the timer was started. There are two parts to solving this issue: updating the display and doing it at regular intervals. Let's first update the display.

1. Create two new properties of type `long` in the `TimerActivity` class called `startedAt` and `lastStopped`:

```
protected long startedAt;  
protected long lastStopped;
```

2. In the `clickedStart` method, set `startedAt` to contain the current time in milliseconds:

```
startedAt = System.currentTimeMillis();
```

And in the `clickedStop` method, set `lastStopped` to contain the current time in milliseconds:

```
lastStopped = System.currentTimeMillis();
```

In this way, you can determine how long the timer has been running between a start click and a stop click.

3. Create a new method called `setTimeDisplay` that sets the counter's text to the elapsed time. The method should look like Listing 4.6.

Listing 4.6 Method for Displaying the Elapsed Time

```
protected void setTimeDisplay() {
    String display;
    long timeNow;
    long diff;
    long seconds;
    long minutes;
    long hours;

    Log.d(CLASS_NAME, "Setting time display");

    if (timerRunning) {
        timeNow = System.currentTimeMillis();
    } else {
        timeNow = lastStopped;
    }

    diff = timeNow - startedAt;

    // no negative time
    if (diff < 0) {
        diff = 0;
    }

    seconds = diff / 1000;
    minutes = seconds / 60;
    hours = minutes / 60;
    seconds = seconds % 60;
    minutes = minutes % 60;

    display = String.format("%d", hours) + ":"
        + String.format("%02d", minutes) + ":"
        + String.format("%02d", seconds);

    counter.setText(display);
}
```

The first section of Listing 4.6, after the local variable declarations and log call, checks to see whether the timer is running. If it is, it gets the current time; otherwise, it gets the time when the Stop button was last clicked.

The difference between the time the Start button was clicked (stored in `startedAt`) and the current time (stored in `timeNow`) is then calculated. This

gives the number of milliseconds that the counter has been running. Make sure that the difference is a positive number. You wouldn't want to display a negative time value.

The time difference is in milliseconds and needs to be converted to a more human-friendly representation of time. From the number of milliseconds, you can calculate the number of seconds, minutes, and hours through integer division and modulo arithmetic (the remainder after a number is divided by another). This reflects the way minutes and seconds normally wrap around on a clock.

Once the time is calculated, you can create and format a time string by using `String.format`. Notice the use of the format `String %02d`, which pads the minutes and seconds with an initial zero if needed.

Then the counter text can set to the value of the time-formatted string stored in `display`.

4. Add a call to the `setTimeDisplay` method at the end of the `clickedStart` and `clickedStop` methods.
5. **Run** the application. Click the Start button, and the timer will display 0:00:00. Wait a few seconds, and then click the Stop button. The timer will now display something different, such as 0:00:03.

Displaying a Running Timer

Next, you need to update the display at regular intervals so that the current time is displayed. On Android this is not as straightforward as it may seem.

The activity's user interface runs in a single thread. If you block that thread for too long, the Android OS thinks your application has frozen, and you will get the Application Not Responding (ANR) dialog box. Strict mode (which you added earlier) will tell you about potential issues that could cause your application to become unresponsive.

One solution is to create an extra thread and do all the work in that thread; in this way, you would not block the main UI thread and would stop any ANRs. Unfortunately, though, simply using standard Java timers or threads is not the answer. That's because the Android SDK is not thread safe, and any thread you create in this manner will not be able to update the display. Only the UI thread can update the display.

The solution? You can create a timer by using the `Runnable` interface and the `Handler` class.

The `Runnable` interface defines a single method called `run` that you implement. (It takes no parameters and returns `void`.) This `run` method is called once when the new thread is started.

The `Handler` class allows you to queue calls to the `run` method (and a few other things) in a `Runnable` class. You can use this class to make a timer that fires at regular intervals.

There are a couple of other ways of implementing this—for instance, using `AsyncTask` or `Services`—but using `Runnable` and `Handler` is the most straightforward way. In the following, you will create a timer using the `Runnable` and `Handler` process.

1. Open `TimerActivity.java`, and, at the top of the class, create a static `long` called `UPDATE_EVERY`. Set it to a value of 200; this is how often you want the screen counter to update. If you set it to 1000, it may not exactly match every second, and the timer display may miss seconds. You might want to play with this value to see what works best.

```
private static long UPDATE_EVERY = 200;
```

2. Create a new class called `UpdateTimer` that implements `Runnable` and has a single `run` method. In the `run` method, log that it has been called.

```
class UpdateTimer implements Runnable {

    public void run() {
        Log.d(CLASS_NAME, "run");
    }
}
```

3. Add a handler property and an `updateTimer` property to the class:

```
protected Handler handler;
protected UpdateTimer updateTimer;
```

Run **Quick fix** to add the import statement for the `Handler` class, making sure it is the `android.os.Handler` class that you import. The `UpdateTimer` class doesn't need an import, because it's in the same package as `Handler`.

4. At the end of the `clickedStart` method in the `TimerActivity` class, create a new instance of both properties, and call the handler's `postDelayed` method. This will cause the `run` method of `UpdateTimer` to be called in 200 milliseconds.

```
handler = new Handler();
updateTimer = new UpdateTimer();
handler.postDelayed(updateTimer, UPDATE_EVERY);
```

5. **Debug** the application. Check that the `run` method is logged when you click the Start button.
6. At the end of `clickedStop`, stop any pending call to the `run` method by calling `removeCallbacks` and set the handler to `null`.

```
handler.removeCallbacks(updateTimer);
handler = null;
```

7. In the `run` method, comment out the log call (otherwise, the `LogCat` view will be flooded with messages). Add calls to set the timer display and call the `run` method again in another 200 milliseconds (via a call to `postDelayed`).

```
setTimeDisplay();  
if (handler != null) {  
    handler.postDelayed(this, UPDATE_EVERY);  
}
```

The null check is to make sure that the handler exists and the Start button has been clicked.

8. **Run** the application again. You should now see the timer counting up when the Start button is pressed, and the timer stopping when the Stop button is pressed (see Figure 4.6).

The application seems as though it is now working. Not quite. Run the application on a USB-connected device, start the timer, wait a while, and rotate the screen. What happened? If you're running in an emulator, you can rotate the screen via **Ctrl + F12** on Windows and **Ctrl + fn + F12** on Mac. The activity lifecycle, discussed briefly in Chapter 3, is the reason the application did not function. In the next section, you will examine the activity lifecycle in more detail to get to the bottom of this.

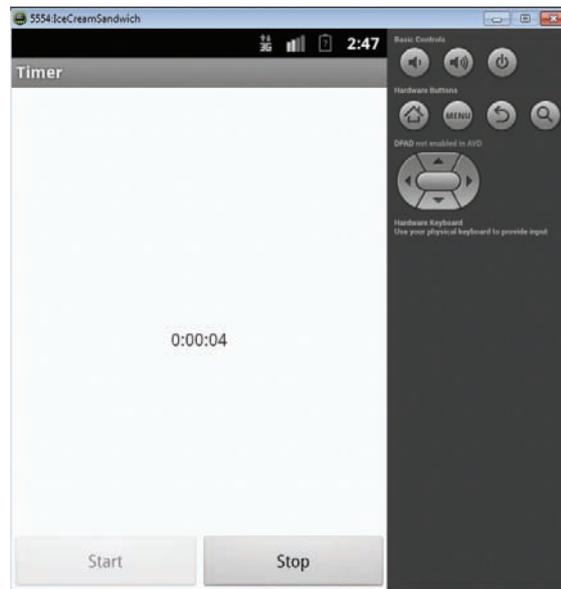


Figure 4.6 Debugging application showing timer

Understanding the Activity Lifecycle

As you have seen, an activity is simply a screen or user interface in an Android application—either a full screen or a floating window that a user interacts with. An Android app is made up of different activities that interact with the user as well as one another. For example, a simple calculator would use one single activity. If you enhanced the calculator app to switch between a simple version and a scientific version, you would then use two activities.

Every Android application runs inside its own process. Processes are started and stopped to run an application and also can be killed to conserve memory and resources. Activities, in turn, are run inside the main UI thread of the application's process.

Once an activity is launched, it goes through a **lifecycle**, a term that refers to the steps the activity progresses through as the user (and OS) interacts with it. There are specific method callbacks that let you react to the changes during the activity lifecycle.

The activity lifecycle has four states.

- When the activity is on the foreground of the application, it is the *running* activity. Only one activity can be in the running state at a given time.
- If the activity loses focus but remains visible (because a smaller activity appears on top), the activity is *paused*.
- If the activity is completely covered by another running activity, the original activity is *stopped*. When an activity stops, you will lose any state and will need to re-create the current state of the user interface when the activity is restarted.
- While the activity is paused or stopped, the system can kill it if it needs to reclaim memory. The user can restart the activity.

While the application moves through the different states, the `android.app.Activity` lifecycle methods (or callbacks) get called by the system. These callbacks are as follows.

- `onCreate(Bundle savedInstanceState)` is called when the activity is created for the first time. You should initialize data, create an initial view, or reclaim the activity's frozen state if previously saved (this is covered later). The `onCreate` callback is always followed by `onStart`.
- `onStart()` is called when the activity is becoming visible. This is an ideal place to write code that affects the UI of the application, such as an event that deals with user interaction. This callback is normally followed by `onResume` but could be followed by `onStop` if the activity becomes hidden.
- `onResume()` is called when the activity is running in the foreground and the user can interact with it. It is followed by `onPause`.

- `onPause()` is called when another activity comes to the foreground. The implementation needs to be quick, because the other activity cannot run until this method returns. The `onPause` callback is followed by `onResume` if the activity returns to the foreground, or by `onStop` if the activity becomes invisible.
- `onStop()` is called when the activity is invisible to the user; either a new activity has started, an existing activity has resumed, or this activity is getting destroyed. The `onStop` callback is followed by `onRestart` if the activity returns to the foreground.
- `onRestart()` is called when the activity is being restarted, as when the activity is returning to the foreground. It is always followed by `onStart`.
- `onDestroy()` is called by the system before the activity is destroyed, either because the activity is finishing or because the system is reclaiming the memory the activity is using.

Figure 4.7 illustrates the various states the activity goes through and the order in which the callback methods get invoked.

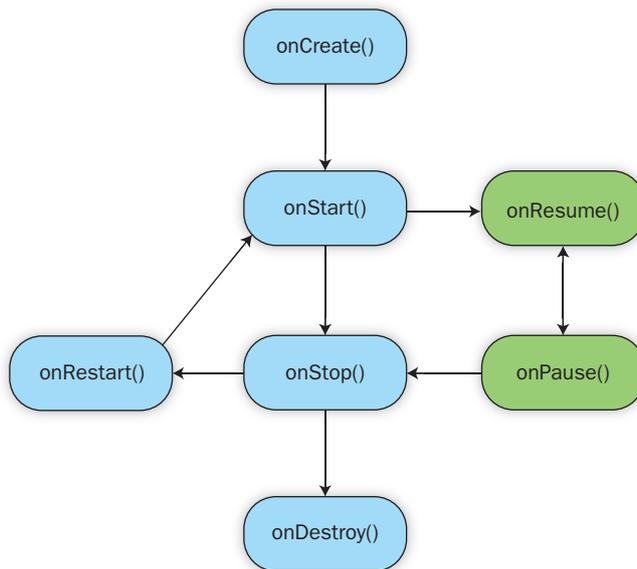


Figure 4.7 Activity lifecycle showing activity states

Exploring the Android Activity Lifecycle

Now let's look at how the Android activity lifecycle works. In Chapter 3, you overrode the `onCreate` method. Now you'll override the remaining lifecycle methods in your `TimerActivity` class by following these steps.

1. Open the **TimerActivity.java** file in the project, and override the existing `onStart` method, which is called when the activity is first viewed. Call the `onStart` method of the parent class, and log a debug message:

```
@Override
public void onStart(){
    super.onStart();
    Log.d(CLASS_NAME, "onStart");
}
```

2. Override the existing `onPause` method, which is called when another activity is called to the foreground. Call the `onPause` method of the parent and log a debug message:

```
@Override
public void onPause(){
    super.onPause();
    Log.d(CLASS_NAME, "onPause");
}
```

3. Override the existing `onResume` method, which is called when the activity is running in the foreground and the user can interact with it. Call the `onResume` method of the parent class, and log a debug message:

```
@Override
public void onResume(){
    super.onResume();
    Log.d(CLASS_NAME, "onResume");
}
```

4. Override the existing `onStop` method, which is called when the activity is invisible to the end user. Call the `onStop` method of the parent class, and log a debug message:

```
@Override
public void onStop(){
    super.onStop();
    Log.d(CLASS_NAME, "onStop");
}
```

- Override the existing `onDestroy` method, which is called when the activity is removed from the system and can no longer be interacted with. Call the `onDestroy` method of the parent class, and log a debug message:

```
@Override
public void onDestroy(){
    super.onDestroy();
    Log.d(CLASS_NAME, "onDestroy");
}
```

- Override the existing `onRestart` method, which is called when the activity is started again and returns to the foreground. Call the `onRestart` method of the parent class and log a debug message:

```
@Override
public void onRestart(){
    super.onRestart();
    Log.d(CLASS_NAME, "onRestart");
}
```

- Now **debug** your application on a device, and look at the debug messages (in the *LogCat* view) that show the changes of state in the application, as shown in Figure 4.8. Experiment with the application to see which state changes occur.

- Turn your device on its side to see if the state changes. The activity is re-created when you do this, and in that process it loses all state.
- Navigate to another application, and see which methods are called.
- Let your device go to sleep, and then unlock the screen to see your application again.



Figure 4.8 LogCat showing activity lifecycle

Fixing Activity Lifecycle Issues

As you've seen, when the application is not running there is no need to have the timer display update, and when the timer activity is re-created you need to refresh the display to put it into the correct state.

To fix these issues you need to update the screen at the correct time.

1. When `onStart` is called and the timer is still running, start calling the `run` method of `UpdateTimer` again. Add this code to the `onStart` method:

```
if (timerRunning) {
    handler = new Handler();
    updateTimer = new UpdateTimer();
    handler.postDelayed(updateTimer, UPDATE _ EVERY);
}
```

2. When `onStop` is called, you no longer need to update the display. Add this code to the `onStop` method:

```
if (timerRunning) {
    handler.removeCallbacks(updateTimer);
    updateTimer = null;
    handler = null;
}
```

3. When `onResume` is called, you need to refresh the display. Add these two lines of code:

```
enableButtons();
setTimeDisplay();
```

4. Debug the application on a device, and rotate the screen when the timer is running. You should now see that the application behaves as you would expect.

Making an Android Device Vibrate

Sometimes a device's screen may not be visible (for example, if it's in someone's pocket), so you need to indicate that time has passed in a nonvisual way. Making the device vibrate is a good way to do this.

Let's set up the code to vibrate once every 5 minutes, twice every 15 minutes, and three times every hour while the timer is running.

1. Add a property called `vibrate` of type `Vibrator` to the `TimerActivity` class:

```
protected Vibrator vibrate;
```

2. Add a property called `lastSeconds` of type `long`. This is needed because the `run` method is called several times a second, and you want the device to vibrate only once.

```
protected long lastSeconds;
```

3. In the `onStart` method, set up the `vibrate` property by calling `getSystemService`. Not all devices can vibrate (and most tablets can't), so you need to check and log when a device doesn't support the feature:

```
vibrate = (Vibrator) getSystemService(VIBRATOR_SERVICE);
```

```
if (vibrate == null) {
    Log.w(CLASS_NAME, "No vibration service exists.");
}
```

4. Add a new method called `vibrateCheck`, which should look like Listing 4.7. This method uses a similar approach as `setTimeDisplay`'s to work out the time difference, but you need only calculate the current minutes and seconds.

To vibrate the device, you call the `vibrate` method, passing it an array of numbers. The numbers represent a vibration pattern, with the first number being the number of milliseconds to wait before starting. This is followed by how long it should vibrate and how long it should pause between each vibration.

Listing 4.7 Method for Vibrating a Number of Times at Regular Intervals

```
protected void vibrateCheck() {
    long timeNow = System.currentTimeMillis();
    long diff = timeNow - startedAt;
    long seconds = diff / 1000;
    long minutes = seconds / 60;

    Log.d(CLASS_NAME, "vibrateCheck");

    seconds = seconds % 60;
    minutes = minutes % 60;

    if (vibrate != null && seconds == 0 && seconds != lastSeconds) {
        long[] once = { 0, 100 };
        long[] twice = { 0, 100, 400, 100 };
        long[] thrice = { 0, 100, 400, 100, 400, 100 };

        // every hour
        if (minutes == 0) {
            Log.i(CLASS_NAME, "Vibrate 3 times");
            vibrate.vibrate(thrice, -1);
        }
    }
}
```

```

    }
    // every 15 minutes
    else if (minutes % 15 == 0) {
        Log.i(CLASS_NAME, "Vibrate 2 time");
        vibrate.vibrate(twice, -1);
    }
    // every 5 minutes
    else if (minutes % 5 == 0) {
        Log.i(CLASS_NAME, "Vibrate once");
        vibrate.vibrate(once, -1);
    }
}

lastSeconds = seconds;
}

```

Once the minutes and seconds have been calculated, the code needs to check whether it is on one of the three vibration boundaries. If it is, it should vibrate the required number of times. Note the check `seconds != lastSeconds`. This makes sure you don't vibrate more than once per second, because this method could be called multiple times in a single second.

5. Inside the `run` method, add a check (before the handler check and `postDelayed` call) to see whether the timer is running and, if it is, to call the `vibrateCheck` method:

```

if (timerRunning) {
    vibrateCheck();
}

```

6. Debug the application in the emulator, and see that `vibrateCheck` is being called in the *LogCat* view.
7. Debug the application via USB debugging. An error will occur. Correct this error by adding the vibrate permission to the Android manifest file just after `<uses-sdk>`:

```

<uses-permission android:name="android.permission.VIBRATE" />

```

Saving User Preferences

Because an activity's state is not saved automatically during its lifecycle, you need to save user preferences so that you can redisplay an activity in the correct state. Let's see how to do that.

Creating a New Activity

Applications often consist of more than one activity. Let's create a new Settings activity to enable and disable vibration and create the best possible experience for the user.

1. Create a new activity called **activity_settings** via the Android New Activity wizard. Select **BlankActivity** as the template, **Settings** as the activity name, and **activity_settings** as the layout file. Type **Settings** as the title.
2. Open the `activity_settings` file. Change the `RelativeLayout` to a `LinearLayout` with a vertical orientation:

```
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
    xmlns:tools=http://schemas.android.com/tools
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
```

3. Add a new checkbox view inside the linear layout. Give it a new id of `vibrate_check`, and set the `layout_width` and `layout_height` to `wrap_content`. Set a resource text to the value `@string/vibrate_checkbox`:

```
<CheckBox
    android:id="@+id/vibrate_check"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/vibrate_checkbox" />
```

4. Add the new resource string `vibrate_checkbox` to the `strings.xml` file:

```
<string name="vibrate_checkbox">Vibrate</string>
```

Showing a New Activity

To show a new activity, you first need to create an intent. Intents, in their simplest form, are a description of an activity that you want to occur. (You can also start activities in other applications, as covered later in the book.)

Next, you'll create a new intent to display the Settings activity.

1. Open the `activity_timer` layout. To launch the new activity, add a new button to the linear layout. Give the button an ID of `settings_button`, and a click handler to call the method `clickedSettings` when the button is pressed:

```
<Button
    android:id="@+id/settings_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
```

```

    android:text="@string/settings_button"
    android:onClick="clickedSettings" />

```

2. Add the new resource string for the Settings button:

```
<string name="settings_button">Settings</string>
```

3. In the `TimerActivity.java` file, add a new `clickedSettings` method:

```

public void clickedSettings(View view) {
    Log.d(CLASS_NAME, "clickedSettings");
}

```

4. **Debug** the application, and check that the `clickedSettings` call is logged in the `LogCat` view. If an RTE occurs, double-check that the `onClick` contains exactly the same method name as the new method just added.
5. In the `clickedSettings` method, create a new `Intent`. Then pass the application context and the class property of the `SettingsActivity`. Run **Quick fix** to add the import statement for the `Intent` class:

```

Intent settingsIntent = new Intent(getApplicationContext(),
    SettingsActivity.class);

```

6. Display the new activity by calling `startActivity`, passing the intent you just created:

```
startActivity(settingsIntent);
```

7. **Run** the application again, and click the Settings button. The setting activity (displaying a checkbox) with a single checkbox will replace the timer activity, as shown in Figure 4.9.

Saving an Application's State

Application state can be stored in many ways, either as static properties stored globally in the application or through the use of the singleton pattern. This pattern is designed to control object creation, limiting the number of objects to one. Because there is only ever one instance of the application class, you can use that to act as a singleton.

Here's how to create a class to save and retrieve the application settings.

1. Create a new Java class called `Settings`. Add a private static (of type `String`) `CLASS_NAME`, and assign the class name in the class constructor:

```

public class Settings {
    private static String CLASS_NAME;

    public Settings() {
        CLASS_NAME = getClass().getName();
    }
}

```

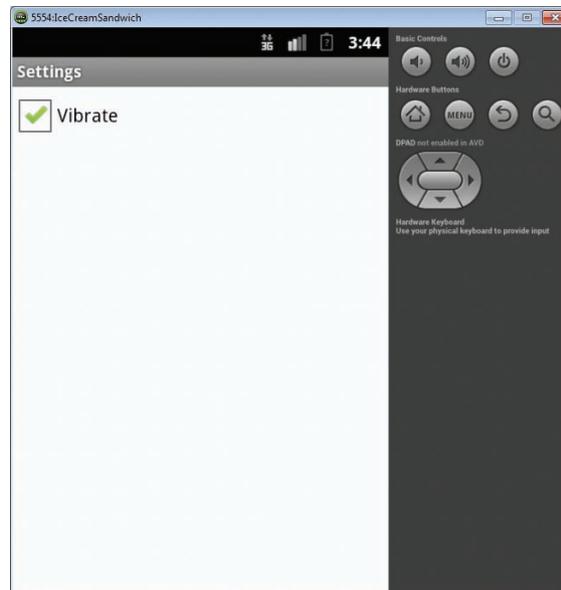


Figure 4.9 The new Settings activity

2. Create a private property to store whether or not the vibrate setting is turned on:

```
protected boolean vibrateOn;
```

3. Create a method to return this property. Run **Quick fix** to import the Log class:

```
public boolean isVibrateOn() {  
    Log.d(CLASS_NAME, "isVibrateOn");  
    return vibrateOn;  
}
```

4. Create a method to set the value of the property:

```
public void setVibrate(boolean vibrate) {  
    Log.d(CLASS_NAME, "setVibrate");  
    vibrateOn = vibrate;  
}
```

5. Create a new class called OnYourBike that extends Application. Add a settings property of type Settings to this class:

```
public class OnYourBike extends Application {  
    protected Settings settings;  
}
```

6. Add a method named `getSettings` that creates an instance of `Settings` if it hasn't already been created, and return the `settings` property:

```
public Settings getSettings() {
    if (settings == null) {
        settings = new Settings();
    }
    return settings;
}
```

7. Add a method named `setSettings` that changes the `settings` property to the `settings` value passed in:

```
public void setSettings(Settings settings) {
    this.settings = settings;
}
```

8. Change the Android manifest file so that the application uses this class as its application by setting the `android:name` attribute to `".OnYourBike"`:

```
<application android:name=".OnYourBike"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
```

9. Open **SettingActivity.java**, and add a `vibrate` checkbox property. Run **Quick fix** to import the `CheckBox` class:

```
private CheckBox vibrate;
```

10. In the `onCreate` method, after the call to `setContentView`, obtain access to the checkbox by calling `findViewById`:

```
vibrate = (CheckBox)
    findViewById(R.id.vibrate_checkbox);
```

11. Obtain the settings by calling the `getSettings` method just created:

```
Settings settings = ((OnYourBike) getApplication()).getSettings();
```

12. Just after that, set the state of the checkbox according to the setting:

```
vibrate.setChecked(settings.isVibrateOn());
```

13. Override the `onStop` method to save the settings:

```
@Override
public void onStop() {
    super.onStop();
    Settings settings = ((OnYourBike) getApplication()).getSettings();
    settings.setVibrate(vibrate.isChecked());
}
```

14. **Run** the application, click the **Settings** button, change the settings checkbox, and press the back button. Go back into the setting activity again by clicking the **Settings** button. The vibrate checkbox should still be ticked.

Notice that there was no need to add a handler to the checkbox for the state to be saved when the activity was stopped. Depending on how the activity is used in your application, you may want to save the setting right away rather than wait until the activity is stopped.

Using Shared Preferences

The settings class you created saves the application's state only while it is running. If the application is stopped and restarted, it won't remember the previous state. To fix that, you need to use shared preferences to save the application's state. Shared preferences allow you to save key value pairs on a device.

You can save the vibration setting—whether it's turned on or off—as a preference:

1. Open **Settings.java**, and add a private static string called `VIBRATE`:

```
private static String VIBRATE = "vibrate";
```

2. In the `isVibrateOn` method, obtain an instance of shared preferences by calling `activity.getPreferences`:

```
SharedPreferences preferences
    = activity.getPreferences(Activity.MODE_PRIVATE);
```

Run **Quick fix** to import the `SharedPreferences` and `Activity` classes.

3. Check whether the `VIBRATE` keys exist, and, if they do, set `vibrateOn` to be the saved value:

```
if (preferences.contains(VIBRATE)) {
    vibrateOn = preferences.getBoolean(VIBRATE, false);
}
```

4. Change the `isVibrateOn` method to take a single parameter of type `Activity`:

```
public boolean isVibrateOn(Activity activity)
```

5. In the `setVibrate` method, after the existing code, save the `vibrate` property by getting access to the shared preferences, creating an editor, saving the property by calling `putBoolean`, and committing the changes by calling `apply`:

```
SharedPreferences preferences
    = activity.getPreferences(Activity.MODE_PRIVATE);
Editor editor = preferences.edit();
editor.putBoolean(VIBRATE, vibrate);
editor.apply();
```

Run **Quick fix** to import the `Editor` class.

6. Change the `setVibrate` method to take an additional parameter of type `Activity`:

```
public void setVibrate(Activity activity, boolean vibrate)
```

7. Open `SettingsActivity.java`, and fix the two errors by passing `this` to the `isVibrateOn` and `setVibrate` methods:

```
vibrate.setChecked(settings.isVibrateOn(this));  
settings.setVibrate(this, vibrate.isChecked());
```

8. **Run** the application, click **Settings**, check the **vibrate** checkbox, and press the **back** button. Click **Menu**, and select **all apps**. Select your application, and click **force stop**. **Run** the application again, and click the **Settings** button. The **vibrate** checkbox should still be checked.

Summary

It's important to refactor and keep your code clean, as you've learned in this chapter. Android gives you a way to lay out child views in relation to each other and to their parent view group, and adding event handlers to your code lets your app react to button clicks.

Looking further into the activity lifecycle identifies a few issues with the application you're building. To fix these issues, you implement simple threading by using the `Runnable` interface and the `Handler` class. (Remember it's important to not hold up the main thread of the UI, or you'll get the dreaded Application Not Responding dialog box.) You can display a new activity by creating an intent and calling `startActivity`.

To store your application's state, you can create a data model and extend the `Application` class. In this data model, you store a simple user preference to control whether or not the device will vibrate.

Index

A

Action bars

- Application icon on, 99
- creating, 97–99
- detecting Android version, 100
- enabling/disabling Go Home icon, 168–169
- Share icon on, 245–246

Activities

- accessing TextView inside, 45–46
- applying styles to, 149
- back button for returning to previous, 92
- converting to fragments, 355–359
- creating, 31, 118
- creating and showing, 75–76
- creating list activity, 209–212
- lifecycle of, 44–45
- main executable file for, 42–44
- maps. *See* map activity
- photographs. *See* photo activity
- returning to home activity, 99–101
- settings. *See* Settings activity
- timers. *See* Timer activity
- trips activity. *See* Trips activity
- understanding Java activity file, 42–44

Activity lifecycle

- adding lifecycle methods, 236–237
- callbacks (methods), 68–69
- exploring how it works, 70–71
- fixing issues with, 72
- states of, 68
- testing, 305–308
- understanding, 44–45

ADB (Android Debug Bridge), 112

adb devices command

- checking running emulators, 316
- showing list of connected devices and emulators, 112

AdMob advertising service, 370–372

ADT (Android Developer Tools)

- design mode, 38
- installing on Mac computers, 26–27
- installing on Windows computers, 21–24
- Lint tool in, 37

Advertising

- employing in applications, 369–372
- for monetizing mobile apps, 11

AirPush advertising service, 370

AlarmManager, checking battery with, 276–278

Amazon Appstore

- marketplaces for Android apps, 363
- overview of, 372–373
- uploading app to, 373–376

Amazon coins, 373

Android applications. *See* Applications

Android Asset Studio

- creating launcher icons, 134–136, 365
- creating notification icons, 136–137

Android Debug Bridge (ADB), 112

Android Developer Tools. *See* ADT (Android Developer Tools)

Android development, introduction

- Android applications and, 10
- building native applications, 2–3
- compared with other mobile OSs, 2
- Google Play and, 10–11
- history of, 3

- Android development, introduction, *continued*
 - market share of Android OS versions, 8
 - overview of, 1
 - summary, 10–11
 - user interface, 8–10
 - versions releases, 3–7
 - Android Device Dashboards, 21
 - Android Device Manager, installing Android application on devices, 36–37
 - Android SDK Manager. *See* SDK Manager
 - Android Studio, as free development IDE, 14
 - Android Support Library
 - installed automatically when creating applications, 31
 - viewing features by version releases, 348
 - Android Virtual Device. *See* AVD (Android Virtual Device)
 - AndroidManifest.xml file, 49–50. *See also* Manifest file
 - ANR (Application Not Responding) dialog box, 65
 - ant, compiling tests with, 317–318
 - APIs (application programming interfaces), in Android SDK, 19–21
 - APK (Android Application Package)
 - compiling applications and installing on devices, 10
 - uploading to Amazon Appstore, 374
 - uploading to Google Play, 365
 - Apkudo testing service, 324
 - Application icon, on action bar, 99
 - Application Manager, uninstalling applications with, 8
 - Application Not Responding (ANR) dialog box, 65
 - Application state
 - saving, 76–79
 - shared preferences, 79–80
 - Applications
 - adding map objects to, 183–186
 - creating first. *See* On Your Bike application, creating
 - customizing. *See* On Your Bike application, customizing
 - introduction to, 10
 - styling. *See* Styling applications
 - testing. *See* Testing applications
 - Arabic
 - language support, 330
 - right-to-left formatting, 344
 - Asian languages, support for, 330
 - Asynchronous tasks, speeding up databases and, 280–284
 - AsyncTask class, 280–284
 - Attributes, styles as, 149
 - Australia, dealing with regional word variations, 343
 - Automatic tests
 - installing Jenkins for, 318–319
 - overview of, 316
 - running from command line, 316–318
 - running with Jenkins, 322–323
 - version control with Git, 319–322
 - AVD (Android Virtual Device)
 - best practices, 34–36
 - cloning existing virtual device, 138–139
 - creating for testing applications, 32–33
 - running application in, 33–34
 - starting Jelly Bean emulator from, 316
- B**
- Back button
 - adding to application, 93–94
 - convention for use of, 92–93
 - Backward compatibility
 - Android Support Library and, 348
 - Android version checking, 349
 - expanding device support by setting SDK at lower level, 349–352
 - overview of, 348
 - Banner ads, 371–372
 - BaseActivity class, Java classes, 174
 - Bash commands, Git, 320–321
 - Battery
 - checking with AlarmManager, 276–278
 - getting current level, 270–272
 - improving life of, 267–268
 - reacting if level is critical, 272
 - Bitmaps
 - displaying in ImageView, 239
 - nine-patch PNG for, 149–150
 - scaling, 131
 - sharing content with friends, 247
 - Blackberry, 1

- Blogs, building media strategy for selling apps, 363–364
- Brightness of screen, Night mode and, 151–153
- Buttons
 - adding photo button, 233
 - adding start and stop buttons for timer, 56–60
 - applying dimension values to resources, 140–141
 - applying styles to, 149–150
 - changing font size, 130–131
 - creating event handlers for, 60–63
 - hiding/disabling camera button, 229–231
 - starting Photo activity from, 234
- C**
- calcLatitude method
 - battery use and, 275
 - changing calculation of latitude and longitude, 191–194
- calcLongitude method
 - battery use and, 275
 - changing calculation of latitude and longitude, 191–194
- Callbacks (methods), activity lifecycle, 68–69
- Camera class, Java classes, 226–227
- Cameras. *See also* Photographs
 - checking if device has camera and photo app, 226–229
 - hiding/disabling camera button, 229–231
 - requirements for distributing apps via Google Play, 368
 - testing activity lifecycle, 305–308
- Cell towers
 - checking accuracy of GPS location, 176–177
 - comparing location awareness methods, 170
 - dealing with location inaccuracies, 190–191
 - finding device location, 169
- checkBattery method
 - calling from onReceive, 277–278
 - debugging, 276
 - overview of, 270–272
- Checkboxes, applying styles to, 149–150
- Chinese, language support, 330
- Chooser control, creating/launching, 242–245
- Classes. *See also* by individual classes
 - adding properties to, 85
 - creating instances of, 88
 - creating methods for, 85–88
 - creating new classes in Eclipse, 84–85, 89–92
- Click handlers, adding to buttons, 60–63
- Code, refactoring. *See* Refactoring code
- Command line
 - checking JRE version, 14
 - checking table creation, 112
 - Jenkins tool operating from, 322–323
 - Monkey tool operating from, 313
 - running tests from, 316–318
- Compatibility, backward. *See* Backward compatibility
- Configuration qualifiers, 132–134
- Constructor, for map activity, 184
- Content ratings, distributing apps via Google Play and, 365–366
- Continuous integration, 318–319
- Coordinates
 - adding to trip coordinate list, 207–208
 - displaying on Google Maps, 218–220
 - getting trip coordinates in proper order, 216–217
 - storing route information in trip database, 203–204
- CPCs (costs per clicks), for advertising, 370–372
- CPMs (cost per thousand impressions), for advertising, 370–372
- CPUs, viewing CPU usage, 268–270
- createTable command, SQL command for creating tables, 115
- Cross-platform frameworks, 2–3
- Cupcake
 - history of Android version releases, 3
 - market share of Android OS versions, 8
- Currencies, localization of apps and, 346–348
- Cursors
 - looping over, 213
 - for moving over a set of rows returned from a query, 120

D

Dalvik Debug Monitor Server (DDMS), 176

Data models, creating, 108

Data types, in SQLite, 109

Databases

- applying trip database, 205–209
- checking table creation, 112–113
- creating, 109–111
- creating data models, 108
- creating relationships between tables, 113–117
- inserting, updating, deleting GPS data, 197–199
- speeding up databases with asynchronous tasks, 280–284
- speeding up databases with indexes, 278–280
- storing GPS data in, 196–197
- storing trip route information in, 201–205

date data type, SQLite, 109

Dates, localization and, 346–348

DDMS (Dalvik Debug Monitor Server), 176

Debug perspective, Eclipse IDE, 18

Debugging

- enabling USB debugging on device, 36–37
- logging inside applications, 46–49
- timer button view, 59–60

delete command, for removing data, 200–201

Density-independent pixel. *See* Dp (density-independent pixel)

Design mode, ADT (Android Developer Tools), 38

Developer account, creating on Google Play, 364

Development environment, setting up

- installing ADT (Android Developer Tools) on Windows, 21–24
- installing Android SDK on Windows, 19–21
- installing Eclipse IDE on Windows, 16–19
- installing Java JDK and JRE on Windows, 14–15
- installing/using Java on Mac computers, 24–27

- overview of, 12–13
- summary, 27
- understanding Java versions, 16

Devices

- adding vibration to, 72–74
- adding WhereAmI class to, 173–175
- Android Device Dashboards, 21
- Android virtual. *See* AVD (Android Virtual Device)
- camera support, 231
- changing layout for tablet devices, 145–146
- checking for camera and photo app, 226–229
- comparing ways of finding device location, 170
- expanding device support by setting SDK at lower level, 349–352
- installing APK on, 10
- installing applications, 36–37
- showing list of connected, 112
- styling applications and, 128–129
- testing applications on range of, 323–325
- testing GPS in virtual device, 175–176
- testing photo app in, 225
- ways for finding device location, 169
- widgets for personalizing, 8–9

Dialect support, localization of apps and, 342

Dimensions, applying dimension values to resources, 140–141

Directories, in Android SDK, 21

Donut

- history of Android version releases, 4
- market share of Android OS versions, 8

Dp (density-independent pixel)

- converting to pixels, 127–128
- definitions of screen-related concepts, 126–127
- for spacing and layout, 129

Drawing, on maps, 187–190

drop command, SQL command for removing tables, 112–113, 117

E**Eclair**

- history of Android version releases, 4
- market share of Android OS versions, 8

- Eclipse IDE (Integrated Development Environment)
 - ADT plug-in, 21–24
 - building Android applications, 30
 - configuring Java JRE in, 17
 - creating classes in, 84–85, 89–92
 - installing on Mac computers, 25
 - installing on Windows computers, 16–17
 - keyboard shortcuts, 19
 - perspectives, 18
 - running tests from, 286
 - views, 18
 - Emulator Control view, in Eclipse, 175–176
 - Emulators
 - best practices for running, 34–35
 - Jelly Bean AVD as, 32–34
 - running tests from command line, 316–318
 - showing list of connected, 112
 - testing GPS in virtual device, 175–176
 - testing photo app in, 225
 - English, dialect support and, 342
 - Event handlers, creating button event handlers, 60–63
- F**
- Files
 - layout files, 38–39
 - reference files, 39–40
 - types of, 37–38
 - understanding activity files, 42–44
 - understanding manifest file, 49–50
 - using IDs with layout files, 40–42
 - Filters, distributing apps via Google Play and, 367
 - Fonts
 - changing size of, 130
 - measuring in sp units, 129
 - Foreign keys
 - creating and dropping tables and, 117
 - speeding up databases with indexes, 279–280
 - turning on foreign key support, 114
 - Formats
 - currency and number formats by region, 346
 - language formats, 344–346
 - Fragments
 - converting activity to, 355–359
 - lifecycle methods, 355
 - overview of, 355
 - using in side-by-side views in tablet landscape orientation, 359–361
 - French
 - improving translation with user help, 335–337
 - translating, 332–334
 - Friends, sharing content with, 242
 - Froyo
 - history of Android version releases, 4
 - knowing Android devices and, 128–129
 - market share of Android OS versions, 8
- G**
- German
 - language support, 330
 - translating, 337–340
 - getLatitude method, 187
 - getLongitude method, 187
 - Gingerbread
 - history of Android version releases, 4–5
 - knowing Android devices and, 128–129
 - market share of Android OS versions, 8
 - market share of Android versions, 324
 - Settings menu in, 96
 - testing applications on, 323
 - Git
 - GUI and Bash commands, 321–322
 - installing, 320–321
 - version control with, 319–322
 - Globalization. *See* Localization of apps
 - Google
 - Android OS championed by, 2
 - history of Android and, 3
 - Google Licensing, 368–369
 - Google Maps
 - adding map objects to application, 183–186
 - configuring application to use, 183
 - displaying coordinates on, 218–220
 - displaying GPS data, 214–215
 - distributing apps via Google Play and, 368
 - placing markers and drawing on maps, 187–190

Google Maps, *continued*
 setting up Google Play services, 181–183
 showing Google Map view, 185

Google Play
 comparing with Amazon Appstore, 373, 375
 employing advertising in application, 370–372
 error message when trying to view Google Maps, 185
 getting application onto, 364–368
 Google Licensing and, 368–369
 international support, 330, 344
 marketplaces for Android apps, 10–11, 363
 search feature, 364
 setting up service, 181–183
 testing service, 324–325

Google Translator Toolkit
 creating Hebrew translation with, 344
 for rough machine translation, 331–332

GPS (Global Positioning System)
 accuracy of, 176–177
 battery use and, 267
 checking to see if enabled, 173
 comparing ways of finding device location, 170
 dealing with inaccuracies of location data, 190–191
 device location awareness and, 169
 displaying GPS data, 206, 209, 214–215
 improving, 178–181
 inserting, updating, deleting GPS data, 197–199
 storing GPS data, 196–197
 testing in virtual device, 175–176

GPS_PROVIDER, 170

Graphical Layout views
 changing layout for tablet devices, 145–146
 changing layout to landscape mode, 144–145
 creating side-by-side views, 146–148
 error checking in, 58

GUI, Git, 320–321. *See also* UI (user interface)

H

Halo Dark theme
 changing themes, 153–157
 default themes, 97–99

Halo Light theme
 changing themes, 153–157
 default themes, 97–99

Halo Light with dark action bars
 changing themes, 153–157
 default themes, 97–99

Handler process, creating timer using, 66–67

Hardware requirements, distributing apps via Google Play and, 367

Hebrew
 language support, 330
 right-to-left formatting, 344

Height
 configuration qualifiers, 132
 making apps look good on different screen sizes, 137–139
 specifying layout height, 55–56

Hello world application. *See* On Your Bike application, creating

Helper classes
 building, 89–92
 using with notifications, 101

Home screen
 in Android user interface, 8
 enabling/disabling Go Home icon on action bar, 168–169
 returning to home activity, 99–101

Honeycomb
 action bar introduced in, 94
 history of Android version releases, 5
 knowing Android devices and, 128–129
 market share of Android OS versions, 8
 Notification.Builder class, 103
 testing applications on devices running, 323

I

I18N (*internationalization*). *See* Localization of apps

Ice Cream Sandwich
 history of Android version releases, 6
 knowing Android devices and, 128–129

- market share of Android OS versions, 8, 324
- testing applications on devices running, 323
- Icelandic, language support, 330
- Icons
 - adding for photo menu, 234
 - Application icon on action bar, 99
 - Go Home icon on action bar, 168–169
 - launcher icons, 134–136
 - notification icons, 136–137
 - Share icon on action bar, 245–246
- IDE (Integrated Development Environment)
 - Android Studio, 14
 - Eclipse. *See* Eclipse IDE (Integrated Development Environment)
- Images, 244. *See also* Bitmaps
- ImageView
 - adding to layouts, 239–240
 - sharing content with friends, 247
- IMEI (International Mobile Equipment Identify) number, 368
- Indexes, speeding up databases, 278–280
- inMobi advertising service, 370
- Input methods, configuration qualifiers, 132
- insert command, SQL
 - inserting GPS data into database, 197–198
 - inserting rows into table, 116–117
 - updating trip model, 200–201
- Instagram, 223
- Installing
 - ADT (Android Developer Tools) on Windows, 21–24
 - Android SDK on Windows, 19–21
 - Android Support Library, 31
 - APK (Android Application Package) on devices, 10
 - applications on devices, 36–37
 - Eclipse IDE on Windows, 16–19
 - Git, 320–321
 - Java JDK and JRE on Windows, 14–15
 - Java on Mac computers, 24–27
 - JDK (Java Development Kit) on Macs, 24
 - Jenkins, 318–319
 - Jenkins plug-in, 322
 - Integrated Development Environment (IDE)
 - Android Studio, 14
 - Eclipse. *See* Eclipse IDE (Integrated Development Environment)
 - IntelliJ IDEA, Android IDE based on, 14
 - International Mobile Equipment Identify (IMEI) number, 368
 - Internationalization (*I18N*). *See* Localization of apps
 - iOS, comparing mobile OSs, 1
- J**
 - Japanese, language support, 330
 - JAR files, for advertising in applications, 370–371
 - Java
 - changing text size, 142–144
 - installing JDK and JRE on Windows, 14–15
 - installing on Mac computers, 24–27
 - versions, 16
 - Java classes
 - BaseActivity class, 174
 - Camera class, 226–227
 - MapActivity class, 208
 - MapFragment class, 359
 - Notify class, 137
 - OnYourBike class, 120
 - RoutesActivity class, 118, 120
 - Settings class, 76, 82
 - SQLiteHelper class, 204
 - TimerActivity class, 225
 - TimerFragment class, 355
 - WhereAmI class, 186, 191, 206
 - Java Development Kit (JDK)
 - installing on Mac computers, 24
 - installing on Windows computers, 14–15
 - Java files
 - types of Android project files, 37
 - understanding activity files, 42–44
 - Java Runtime Environment (JRE)
 - configuring in Eclipse, 17
 - installing on Windows, 14–15
 - javac-version command, for checking Java compiler version, 24

java-version command, for checking JRE version, 16

JDK (Java Development Kit)
installing on Mac computers, 24
installing on Windows computers, 14–15

Jelly Bean
history of Android version releases, 6–7
knowing Android devices and, 128–129
language support, 330
market share of Android OS versions, 8, 324
notifications in, 101
selecting SDK when building Android application, 30
starting Jelly Bean emulator from AVD, 316
testing applications on devices running, 323

Jenkins
installing, 318–319
running tests with, 322–323

JPG format, 364–365

JRE (Java Runtime Environment)
configuring in Eclipse, 17
installing on Windows, 14–15

JUnit
comparing versions 3 and 4, 299–300
creating test application, 286–291
improving tests by refactoring, 297–299
increasing test coverage, 292–293
running test, fixing failed tests, and re-running, 291–292
speeding up tests, 294–297
testing applications with, 286

JUnit extensions
testing Android applications, 299–300
testing timer activity initial state, 300–302

K

Key value pairs, saving application state and, 79–80

Keyboard shortcuts, Eclipse IDE, 19

Keypads, configuration qualifiers, 132

Killed state, of activity lifecycle, 68

Kindle Fire/Kindle Fire HD/Kindle Fire HDX
accessing Amazon Appstore, 372
developing for, 374–375
uploading app to Amazon Appstore, 373

Kit-Kat, history of Android version releases, 7

L

Landscape orientation
changing layout for, 144–145
handling orientation changes, 251–252
setting in relative layout, 146
side-by-side views in tablets, 359–361

Languages
adding language region codes, 342–343
configuration qualifiers, 132
dealing with regional word variations, 343–344
dialect support, 342
Google Translator Toolkit, 331–332
handling language formats, 344
improving translation with user help, 335–337
right-to-left layouts, 344–346
support, 330–331
translating French, 332–334
translating German, 337–340
translating Spanish, 340–342

Latitude
battery use and, 275
changing calculation of, 191–194
comparing raw location values and corrected location values, 194–196
default location for 0 latitude, 186
getLatitude method, 187
logging accuracy of location data, 172

Launcher icons, 134–136, 364

Layout files
using IDs with, 40–42
XML files, 38–39

Layouts
adding ImageView, 239–240
changing for landscape mode, 144–145
changing for tablet devices, 145–146
creating for photo activity, 232

- creating side-by-side views, 146–148
 - dp (density-independent pixel) for, 129
 - handling orientation changes, 251–252
 - ldrtl/ldltr for layout direction, 132
 - linear layout view, 56–60, 224–226
 - relative layout view, 55–56, 137–139, 146
 - specifying width and height, 55–56
 - supporting right-to-left layouts, 330, 344–346
 - taking photographs and, 224–226
- ldrtl/ldltr, for layout direction, 132
- Licenses, Google Licensing, 368–369
- Light sensors
 - dealing with erratic sensor values, 160–162
 - detecting light levels, 158–160
- Linear layout views
 - adding start and stop buttons for timer, 56–60
 - positioning vertically, 56
 - taking photographs and, 224–226
- Lint, finding errors with, 37
- List activity, creating list activity for trips, 209–212
- ListView
 - converting trip activity to, 211
 - creating for routes, 118–122
- Localization of apps
 - adding language region codes, 342–343
 - backward compatibility and. *See* Backward compatibility
 - building for various screen sizes, 352–354
 - dealing with regional word variations, 343–344
 - dialect support, 342
 - fragments for customizing apps for different countries, 355–361
 - Google Translator Toolkit and, 331–332
 - handling language formats, 344
 - handling variations in dates, numbers, and currencies, 346–348
 - improving translations with user help, 335–337
 - international demand for Android platform, 329–330
 - language support, 330–331
 - overview of, 327
 - refactoring code and, 327–329
 - summary, 361
 - supporting right-to-left layouts, 344–346
 - translating French, 332–334
 - translating German, 337–340
 - translating Spanish, 340–342
- Location awareness
 - accuracy of GPS locations, 176–177
 - adding map objects to application, 183–186
 - adding WhereAmI class to device, 173–175
 - applying trip database, 205–209
 - changing calculation of latitude and longitude, 191–194
 - comparing cell tower or Wi-Fi hotspots with GPS, 170
 - comparing raw location values and corrected location values, 194–196
 - configuring application to use Google Maps, 183
 - creating list activity for trips, 209–212
 - creating WhereAmI class, 170–173
 - dealing with inaccuracies, 190–191
 - displaying current location, 186–187
 - displaying GPS data, 209
 - displaying GPS data in Google Maps, 214–215
 - displaying trip coordinates on Google Maps, 218–220
 - displaying trips activities, 212–214
 - getting trip coordinates in proper order, 216–217
 - improving GPS, 178–181
 - inserting, updating, deleting GPS data, 197–199
 - options for displaying maps activities, 217–218
 - overview of, 165
 - placing markers and drawing on maps, 187–190
 - refactoring code, 165–169
 - setting up Google Play services, 181–183
 - storing GPS data, 196–197
 - storing route information, 201–205
 - summary, 218–220
 - testing GPS in virtual devices, 175–176

- Location awareness, *continued*
 - updating trip model, 200–201
 - ways for finding device location, 169
- Location services
 - correcting issues with, 261–267
 - turning off GPS Satellites in, 180
 - turning on, 179
- LocationListener interface, 171
- LogCat
 - of activity lifecycle, 71
 - logging inside applications, 46–49
 - running application in AVD and, 33–34
- Logging
 - changes to GPS status, 178
 - location-related, 172
 - LogCat view and, 33–34
 - using inside applications, 46–49
- Longitude
 - battery use and, 275
 - changing calculation of, 191–194
 - comparing raw location values and corrected location values, 194–196
 - default location for 0 longitude, 186
 - getLongitude method, 187
 - logging accuracy of location data, 172
- M**
- Mac OSs
 - installing ADT on, 26–27
 - installing Android SDK on, 25–26
 - installing Eclipse IDE on, 25
 - installing JDK on, 24
- MainActivity class, Java classes, 42–44
- Manifest file
 - handling orientation changes, 251–252
 - obtaining device location and, 170
 - refactored activities in, 83
 - refactoring code for social network app, 223
 - themes and, 97
 - understanding, 49–50
- Many to many relationships, between tables, 113
- map activity, 187–190
 - adding map objects to application, 183–186
 - adding menu item for, 173–174
 - checking accuracy of GPS location, 176–177
 - constructor for, 184
 - placing markers and drawing on maps, 187–190
 - showing and starting, 174
- MapActivity class, Java classes, 208
- MapFragment class, Java classes, 359
- Markers, placing on maps, 187–190
- Marketing applications. *See* Selling applications
- MCC (mobile country code), 132
- Media strategy, for selling application, 363–364
- Menus
 - action bar replacing menu-style navigation, 94
 - adding item for taking photos, 233
 - adding items to, 118
 - creating, 95–96
 - creating for trip activity, 211
 - for sharing content, 244
 - starting PhotoActivity from, 233–234
- Methods
 - adding lifecycle methods, 236–237
 - callbacks (methods) of activity lifecycle, 68–69
 - for checking battery, 276–278
 - correcting issues with vibrate and notification methods, 252–253
 - fixing issues with activity lifecycle, 72
 - for formatting, 105
 - for input, 132
 - for latitude and longitude, 187, 191–194
 - modifying, 274–275
 - for new class, 85–88
 - overriding, 70–71, 99, 255
 - for simple menu, 95
- Middle Eastern languages, 330
- MNC (mobile network code), 132
- Mobile country code (MCC), 132
- Mock applications
 - creating, 302–305
 - testing timer activity, 309
- Monkey
 - overview of, 313
 - testing applications, 313–315

N

Native applications

- advantages of, 1
- building, 2–3

Navigation

- action bar replacing menu-style navigation, 94
- configuration qualifiers, 132

Navigator view, Eclipse IDE, 18

NETWORK_PROVIDER, 170

Night mode

- changing themes and, 153–157
- configuration qualifiers, 132
- dealing with erratic sensor values, 160–162
- detecting light levels, 158–160
- enabling, 151–153

Notification class, 101

notification methods, 252–253

Notifications

- creating, 101–103
- creating notification icons, 136–137
- notifyCheck method, 252–253
- options when using, 104
- showing at regular intervals on, 104–107
- what it consists of, 101

Notify class, Java classes, 137

Numbers, localization and, 346–348

O

On Your Bike application, creating

- accessing TextView inside an activity, 45–46
- best practices, 34–36
- creating AVD for testing, 32–33
- file types, 37–38
- installing on devices, 36–37
- layout XML files and, 38–39
- logging, 46–49
- overview of, 29
- reference XML files and, 39–40
- running in AVD, 33–34
- steps in, 30–31
- summary, 50
- understanding activity files, 42–44
- understanding activity lifecycles, 44–45
- understanding manifest file, 49–50
- using IDs with layout files, 40–42

On Your Bike application, customizing

- back button for returning from settings activity, 92–94
- building Toast class and related helper class, 89–92
- checking table creation, 112–113
- creating a toast, 88
- creating action bars, 97–99
- creating data model, 108
- creating database and tables, 109–111
- creating menus, 95–96
- creating notifications, 101–103
- creating relationships between tables, 113–117
- creating routes ListView, 118–122
- improving SettingsActivity, 88
- overview of, 81
- refactoring code and, 82–88
- returning to home activity, 99–101
- showing notifications at regular intervals, 104–107
- summary, 118–122

On Your Bike application, getting onto

- Google Play, 364–368

onClick event handler

- adding to buttons, 62–63
- adding to Toast class, 91

onCreate method

- activity lifecycle and, 68–69
- overriding, 255

onDestroy method

- activity lifecycle and, 68–69
- overriding, 71

One to many relationships, between tables,

113

One to one relationships, between tables,

113

onPause method

- activity lifecycle and, 68–69
- overriding, 70

onRestart method

- activity lifecycle and, 68–69
- overriding, 71

onResume method

- activity lifecycle and, 68–69
- fixing issues with activity lifecycle, 72
- overriding, 70

- onStart method
 - activity lifecycle and, 68–69
 - fixing issues with activity lifecycle, 72
 - overriding, 70
- onStartCommand method, 255
- onStop method
 - activity lifecycle and, 68–69
 - fixing issues with activity lifecycle, 72
 - overriding, 70
- OnYourBike class, Java classes, 120
- Open Handset Alliance, 2–3
- OpenGL ES 2, required for Google Maps, 183
- Optimization
 - checking battery with AlarmManager, 276–278
 - correcting issues with location services, 261–267
 - correcting issues with vibrate and notification methods, 252–253
 - creating services, 254–257
 - determining power usage, 268–270
 - handling orientation changes, 251–252
 - improving battery life, 267–268
 - overview of, 249
 - responding to power levels, 270–276
 - running application as a service, 250–251
 - speeding up databases with asynchronous tasks, 280–284
 - speeding up databases with indexes, 278–280
 - summary, 284
- Orientation
 - configuration qualifiers, 132
 - definitions of screen-related concepts, 126–127
 - handling orientation changes, 251–252
 - setting landscape orientation in relative layout, 146
- OSs (operating systems)
 - ANR (Application Not Responding) dialog, 65
 - comparing Android with other mobile OSs, 2
 - Mac OSs. *See* Mac OSs
 - market share of Android OS versions, 324
 - Windows OSs. *See* Windows OSs
- Overriding methods, 43–44

P

- Package Explorer view, 18
- PASSIVE_PROVIDER, 170
- Paused state, of activity lifecycle, 68
- Permissions, obtaining device location and, 170
- Perspectives, Eclipse IDE, 18
- Photo activity
 - adding activity lifecycle methods to, 236–237
 - adding takePhoto method, 237
 - creating, 232–234
 - displaying photos, 237–241
 - displaying screen for, 235
 - fixing bug in, 315
 - opening PhotoActivity class, 234–237
 - starting from button, 234
 - starting from menu, 233–234
 - testing activity lifecycle, 305–308
 - testing with Monkey, 314–315
- Photographs
 - checking if device has camera and photo app, 226–229
 - creating photo activity, 232–234
 - displaying, 237–241
 - hiding/disabling camera button, 229–231
 - opening PhotoActivity class, 234–237
 - sharing content with friends, 245–247
 - taking, 224–226
- Pixels, converting dp to, 127–128
- Platform version, configuration qualifiers, 133
- Platforms, cross-platform frameworks, 2–3
- PNG format
 - creating launcher icons, 364–365
 - distributing apps via Google Play, 364–365
 - nine-patch PNG for bitmaps, 149–150
- polyline, drawing on maps, 189
- Pop-ups
 - building Toast class and related helper class, 89–92
 - creating a toast pop-up, 88
- Portrait mode, 251–252
- Power usage
 - determining, 268–270
 - responding to power levels, 270–276

- Preferences
 - saving user preferences, 74
 - shared preferences for saving application state, 79–80
 - Primary keys, speeding up databases with indexes, 279–280
 - Promotional graphics, 365
 - Properties, adding to new class, 85
 - Providers, types of, 170
 - Public relations, in media strategy, 364
- Q**
- Quick fix, 45–46
- R**
- r value, for locale settings (language and region), 132
 - Ranking algorithm, in Google Play, 11
 - Refactoring code
 - for application styling exercise, 126
 - creating user interface and, 51–54
 - customizing On Your Bike application, 82–88
 - for localization exercise, 327–329
 - for location awareness activities, 165–169
 - for optimization exercise, 249–250
 - for social networking app, 223
 - for speeding up application testing, 297–299
 - for testing applications project, 285–286
 - Reference files, XML files, 39–40
 - Regions. *See also* Languages; Localization of apps
 - adding language region codes, 342–343
 - configuration qualifiers, 132
 - dealing with regional word variations, 343–344
 - dialect support and, 342
 - handling variations in dates, numbers, and currencies, 346–348
 - list of region codes, 331
 - Relationships, between tables, 113–117
 - Relative layout view
 - making apps look good on different screen sizes, 137–139
 - positioning views, 55
 - setting landscape orientation, 146
 - specifying layout width and height, 55–56
 - Research in Motion (RIM), 1
 - Resolution
 - definitions of screen-related concepts, 126–127
 - distributing apps via Google Play and, 365
 - making applications resolution independent, 129–131
 - Resource qualifiers
 - for language and regions, 330, 343
 - in resource XML file, 361
 - for screen sizes, 140, 352–353
 - Resources
 - adding suffixes to resource directory, 132
 - applying dimension values to, 140–141
 - Right-to-left layouts
 - handling language formats, 344–346
 - language support and, 330
 - RIM (Research in Motion), 1
 - Routes
 - creating dummy, 209
 - creating list activity for trips, 209–212
 - creating ListView, 118–122
 - displaying trips activity, 212
 - inserting, updating, deleting GPS data, 197–199
 - storing route information in trip database, 201–205
 - RoutesActivity class, Java classes, 120
 - Runnable interface, creating timer using, 66–67
 - Running state, of activity lifecycle, 68
- S**
- Samsung testing labs, 324
 - Scale-independent pixels (Sp), for fonts, 129
 - Screen aspect, configuration qualifiers, 132–133
 - Screen density
 - definitions of screen-related concepts, 126–127
 - grouping by dpi, 127–128
 - launcher icons and, 134–136
 - notification icons and, 136
 - Screen size
 - applying dimension values to resources, 140–141
 - configuration qualifiers, 132

- Screen size, *continued*
 - definitions of screen-related concepts, 126–127
 - grouping into categories, 127
 - international support and, 352–354
 - knowing Android devices and, 128–129
 - testing applications on devices, 323
 - variables in Android application design, 10
- Screens
 - battery use and, 267
 - differences in, 126–127
 - enabling night mode, 151–153
 - making apps look good on different screen sizes, 137–139
 - sizes and densities, 127–128
 - testing applications on devices, 323
- SDK (Software Development Kit)
 - Android SDK not thread safe, 65
 - device camera support, 231
 - expanding device support by setting SDK at lower level, 349–352
 - installing on Mac computers, 25–26
 - installing on Windows computers, 19–21
 - selecting when building Android application, 30
 - versions releases, 3–7
- SDK Manager
 - downloading and installing Android SDK versions, 351
 - implementing Google Licensing, 368–369
 - installing Google Play services, 181
 - updating Android SDK on Mac, 25–26
 - updating Android SDK on Windows, 19–22
 - uploading app to Amazon Appstore, 374
- Searches, 11, 274–275
- select command, for showing rows and fields in tables, 116–117, 119–120
- Selling applications
 - Amazon Appstore, 372–376
 - building media strategy, 363–364
 - employing advertising in application, 369–372
 - Google Licensing and, 368–369
 - Google Play services, 364–368
 - monetizing mobile apps via advertising, 11
 - overview of, 363
 - summary, 372–376
- Sensor values, 160–162
- ServiceConnection class
 - correcting issue with location services, 263–267
 - creating services, 254–257
- Services
 - adding binding class to, 256
 - correcting issues with timers, 257–261
 - creating, 254–257
 - running application as, 250–251
 - testing, 310–313
- Settings activity
 - back button for returning from, 92–94
 - displaying, 75–76
 - extending BaseActivity class, 169
 - improving, 88–92
 - overriding methods of, 99
 - refactoring code for customization exercise, 82–83
- Settings class, Java classes, 76, 82
- Settings menu, in Gingerbread, 96
- Share class
 - adding menus, 244
 - adding text and images, 243
 - building, 242–243
- Shared preferences, for saving application state, 79–80
- Sharing content
 - creating/launching chooser control, 242–245
 - with friends, 242
 - text and photos, 245–247
- Side-by-side views
 - creating, 146–148
 - in tablet landscape orientation, 359–361
- Singleton pattern, 76
- Smart phones
 - popularity of Android, 1
 - testing applications on, 324
- Snapshots, creating AVDs and, 32–33
- Social media, building media strategy for selling apps, 363

- Social networking
 - checking if device has camera and photo app, 226–229
 - creating photo activity, 232–234
 - creating/launching chooser control, 242–245
 - displaying photos, 237–241
 - hiding/disabling camera button, 229–231
 - opening PhotoActivity class, 234–237
 - refactoring code and, 223
 - sharing content with friends, 242
 - sharing text and photos, 245–247
 - summary, 248
 - taking photographs, 224–226
- Software requirements, distributing apps via Google Play and, 367
- Sp (scale-independent pixels), for fonts, 129
- Spanish language support, 340–342
- SQL (Structured Query Language), 107
- SQLite database
 - Android support for, 107
 - applying trip database, 205–209
 - creating relationships between tables, 113–117
 - data types in, 109
 - delete command, 200–201
 - drop command, 112–113, 117
 - insert command, 116–117, 197–198, 200–201
 - inserting, updating, deleting GPS data, 197–199
 - select command, 119–120
 - speeding up with asynchronous tasks, 280–284
 - speeding up with indexes, 278–280
 - storing GPS data in, 196–197
 - storing route information in trip database, 201–205
 - update command, 200–201
- SQLiteHelper class, Java classes, 204
- SQLiteOpenHelper class
 - creating database with, 109
 - overview of, 107
- startSearching method, 274–275
- startTimer method, 257
- State
 - saving application state, 76–79
 - shared preferences for saving application state, 79–80
 - testing timer activity initial state, 300–302
- Static properties, saving state and, 76–77
- Status bar, in Android user interface, 9
- Stopped state, of activity lifecycle, 68
- stopTimer method, 257
- Strict mode
 - actions, 54
 - setting up in application, 54–55
- String format method, 105
- Structured Query Language. *See* SQL (Structured Query Language)
- Styles
 - applying to buttons and checkboxes, 150–151
 - overview of, 149
- Styling applications
 - applying dimension values to resources, 140–141
 - applying styles and themes, 149–151
 - changing layout for landscape mode, 144–145
 - changing layout for tablet devices, 145–146
 - changing text size, 142–144
 - changing themes, 153–157
 - configuration qualifiers, 132–134
 - creating launcher icons, 134–136
 - creating notification icons, 136–137
 - creating side-by-side views, 146–148
 - dealing with erratic sensor values, 160–162
 - detecting light levels, 158–160
 - enabling night mode, 151–153
 - knowing Android devices and, 128–129
 - making apps look good on different screen sizes, 137–139
 - overview of, 125
 - refactoring in preparation for, 126
 - resolution independence and, 129–131
 - screen differences and, 126–127
 - screen sizes and densities, 127–128
 - summary, 162–163

T

- Tables
 - checking, 112–113, 207
 - creating, 109–111
 - relationships between, 113–117
- Tablets
 - changing layout for, 145–146
 - side-by-side views in tablet landscape orientation, 359–361
 - testing applications on, 323
- Task Manager, viewing CPU usage, 268–270
- Test app
 - creating, 286–291
 - running test, fixing failed tests, and re-running, 291–292
- TestDrive feature, Amazon Appstore, 373
- Testing applications
 - with Android JUnit extensions, 299–300
 - automatically, 316
 - from command line, 316–318
 - creating AVD for, 32–33
 - creating mock application for testing activities, 302–305
 - creating test app with JUnit, 286–291
 - improving tests by refactoring, 297–299
 - increasing test coverage, 292–293
 - with Jenkins, 318–319, 322–323
 - with JUnit, 286
 - with Lint, 37
 - with Monkey, 313–315
 - on range of devices, 323–325
 - refactoring code for test project, 285–286
 - running test, fixing failed tests, and re-running, 291–292
 - speeding up tests, 294–297
 - summary, 325
 - testing activity lifecycle, 305–308
 - testing by interacting with UI, 309–310
 - testing photo app in emulators and in devices, 225
 - testing timer activity initial state, 300–302
 - testing timer activity with timer running, 308–309
 - version control with Git, 319–322
- Testing services
 - overview of, 310–311
 - testing applications on devices, 324–325
 - testing GPS in virtual device, 175–176
 - testing if running, 311–312
 - testing start-up and shut-down, 312–313
- Text
 - adding to Share class, 243
 - applying dimension values to resources, 140–141
 - changing size in Java, 142–144
 - sharing content with friends, 245–247
- TextView, 45–46
- Thai, 330
- Themes
 - applying, 149–151
 - changing, 153–157
 - for consistent style, 97
 - styles compared with, 149
- Threads
 - activities running inside main UI thread, 68
 - Android SDK not thread safe, 65
 - avoiding database access on main UI thread, 280
 - detecting issues with UI thread, 54–55
 - pausing with sleep method, 291
- Timer activity
 - adding start and stop buttons, 56–60
 - displaying running timer, 65–67
 - increasing test coverage, 292–293
 - refactoring as means of speeding up tests, 297–299
 - speeding up tests, 294–297
 - testing, 286–292
 - testing by interacting with UI, 309–310
 - testing initial state, 300–302
 - testing with timer running, 308–309
 - updating display, 63–65
- Timer activity, optimizing
 - correcting issues with, 257–261
 - correcting issues with vibrate and notification methods, 252–253
 - creating services, 254–257
 - handling orientation changes, 251–252
 - running application as a service, 250–251
- TimerActivity class, Java classes, 225

- TimerFragment class, Java classes, 355
 - TimerService class, 310–313
 - TimerState class, 294
 - Toast pop-up
 - building Toast class and related helper class, 89–92
 - creating, 88
 - displaying, 90–91
 - Touches, for returning to Home screen, 99
 - Touchscreen, configuration qualifiers, 132
 - Translation
 - French, 332–334
 - German, 337–340
 - Google Translator Toolkit for machine translation, 331–332
 - improving with user help, 335–337
 - Spanish, 340–342
 - Trips activity
 - applying trip database, 205–209
 - creating list activity for trips, 209–212
 - displaying, 212–214
 - displaying coordinates on Google Maps, 218–220
 - displaying GPS data, 209, 214–215
 - getting trip coordinates in proper order, 216–217
 - inserting, updating, deleting routes, 197–199
 - options for displaying, 217–218
 - storing route information in trip database, 201–205
 - updating trip model, 200–201
- U**
- UI (user interface)
 - activity lifecycle and, 68–69
 - creating, 51
 - creating and showing new activity, 75–76
 - creating button event handlers, 60–63
 - displaying running timer, 65–67
 - exploring how activity lifecycle works, 70–71
 - fixing activity lifecycle issues, 72
 - indicating relative layout of views, 55
 - linear layout views, 56–60
 - overview of, 8–10
 - refactoring code and, 51–54
 - specifying layout width and height, 55–56
 - summary, 79–80
 - testing applications from, 309–310
 - updating timer display, 63–65
 - update command, updating trip model, 200–201
 - USB, enabling USB debugging on device, 36–37
 - User interface. *See* UI (user interface)
 - Users, improving language translation with user help, 335–337
- V**
- Vector images, vs. bitmaps, 131
 - Version checking
 - backward compatibility and, 349
 - viewing features by version releases, 348
 - Version control
 - Git GUI and Bash commands, 321–322
 - installing Git for, 320–321
 - overview of, 319–320
 - Vibration, making devices vibrate, 72–74
 - Views, Android UI controls
 - applying styles to, 149
 - creating side-by-side views, 146–148
 - graphical layout. *See* Graphical Layout views
 - linear layout. *See* Linear layout views
 - relative layout, 55
 - side-by-side views in tablets, 359–361
 - specifying layout width and height, 55–56
 - Views, Eclipse
 - ImageView, 239–240, 247
 - ListView, 118–122, 211
 - LogCat view. *See* LogCat
 - TextView, 45–46
 - using, 18
 - Virtual devices, 175–176. *See also* AVD (Android Virtual Device)
 - VServ.mobi, 370
- W**
- Welsh, language support, 330
 - WhereAmI class, Java classes, 186, 191, 206
 - adding to device, 173–175
 - correcting issue with location services, 261

WhereAmI class, Java classes, *continued*

creating, 170–173

power usage options, 273

Widgets, personalizing Android devices, 8–9

Width

configuration qualifiers, 132

making apps look good on different
screen sizes, 137–139

specifying layout width, 55–56

Wi-Fi

checking accuracy of GPS location and,
176–177

comparing ways of finding device loca-
tion, 170

dealing with inaccuracies of location data,
191

ways for finding device location, 169

Windows OSs

installing ADT on, 21–24

installing Android SDK on, 19–21

installing Eclipse IDE on, 16–19

installing Java JDK and JRE on,
14–15

X

XML

for action bar, 97

layout files, 38–39

reference files, 39–40

for simple menu, 95

types of Android project files, 37–38

understanding manifest file, 49–50

using IDs with layout files, 40–42

xUnit-testing, 286

This page begins the continuation of the copyright page.

The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.
<http://creativecommons.org/licenses/by/3.0/>

Some figures that appear in this book have been reproduced from or are modifications based on work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution.
<http://creativecommons.org/licenses/by/3.0/>

Portions of this book are modifications based on work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.
<http://creativecommons.org/licenses/by/2.5/>

The “On Your Bike” code and application is copyright 2013 Pearson Education, Inc., and is licensed under the Apache License, Version 2.0 (the “License”); you may not use these files except in compliance with the License. You may obtain a copy of the License at:
<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This book includes software from The Android Open Source Project Copyright © 2005–2008, The Android Open Source Project Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. Apache License Version 2.0, January 2004.
<http://www.apache.org/licenses/>

All art generated by the Android Asset Studio is licensed under a Creative Commons Attribution 3.0 Unported License.
<http://creativecommons.org/licenses/by/3.0/>

Trademark Notices

Google and the Google logo are registered trademarks of Google Inc., used with permission.

Android is a trademark of Google Inc.

The AdMob™ mobile advertising service, Dalvik™ virtual machine, Glass™ wearable computing device, Google Analytics™ web analytics service, Google Checkout™ payment and billing service, Google Maps™ mapping service, Google Translator Toolkit™ tools, Google Wallet™ payment service, Google+™ social service, Open Handset Alliance™ business alliance, Picasa™ photo organizing software, YouTube™ video community are all trademarks of Google Inc.

Google Play is a trademark of Google Inc.

The Nexus One™ mobile phone, Nexus S™ mobile phone, and Nexus™ family of marks for mobile devices and peripherals are all trademarks of Google Inc.

Apache is trademark of The Apache Software Foundation. Used with permission. No endorsement by The Apache Software Foundation is implied by the use of these marks.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates.

Apple, Mac, and OS X are trademarks of Apple Inc., registered in the U.S. and other countries.

Windows, Windows XP, Windows Vista, Windows 7, and Exchange are registered trademarks of Microsoft Corporation in the United States and other countries.

“Eclipse” is a trademark of Eclipse Foundation, Inc.

IntelliJ® is a registered trademark owned by JetBrains s.r.o.

SQLite is a registered trademark owned by Hwaci.

OpenGL is a registered trademark of Silicon Graphics International Corp. in the U.S. and/or other countries worldwide.

Apache Ant, Apache Subversion, and Subversion are trademarks of The Apache Software Foundation.

GitHub is a trademark of Github, LLC.

SourceTree by Atlassian is a trademark of Atlassian.

Twitter is a registered trademark of Twitter, Inc.

Facebook® and Instagram™ are trademarks and registered trademarks of Facebook Inc.

Apkudo is a trademark of Apkudo LLC.

Adobe and Flash are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

PhoneGap is a trademark of Adobe Systems Incorporated.

Sencha Touch™ are registered trademarks of Sencha, Inc.

Titanium™ is a trademark of Appcelerator, Inc., registered in the U.S. and /or other countries and is used under license.

Kendo UI is a trademark of Telerik AD.

All of the Tumblr trademarks displayed in this book are the property of Tumblr, Inc.

Amazon, Amazon Appstore for Android, Kindle, and Kindle Fire are trademarks of Amazon.com, Inc., or its affiliates.

BlackBerry®, RIM®, Research In Motion® and related trademarks, names, and logos are the property of Research In Motion Limited and are registered and/or used in the U.S. and countries around the world. Used under license from Research In Motion Limited.

MOTOROLA and MOTOBUR are trademarks or registered trademarks of Motorola Trademark Holdings, LLC.

HTC is a trademark of HTC Corporation.

LG is a registered trademark of LG Electronics, Inc.

Samsung, Galaxy S3, Galaxy S4, Galaxy Note, Galaxy Camera, Pocket, and TouchWiz are trademarks of Samsung Electronics Co., Ltd.

OUYA is a trademark of OUYA, Inc.

Verizon Wireless is a trademark of Verizon Trademark Services, LLC.

AT&T is a trademark of AT&T Intellectual Property or AT&T affiliated company (“AT&T Marks”).

T-Mobile is registered and/or unregistered trademark of Deutsche Telekom AG.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc.

Wi-Fi® is a registered trademark of Wi-Fi Alliance.

USB is a trademark or registered trademark of USB Implementers Forum corporation in the United States and/or other countries.

SD is a trademark or registered trademark of SD-3C, LLC, in the United States, other countries, or both.