



DEITEL® DEVELOPER SERIES

THIRD EDITION

Android™ 6

8 Fully Coded
Android™ Apps

for Programmers

An App-Driven Approach



PAUL DEITEL • HARVEY DEITEL
ALEXANDER WALD

FREE SAMPLE CHAPTER



SHARE WITH OTHERS



**ANDROID™ 6 FOR
PROGRAMMERS**
AN APP-DRIVEN APPROACH
THIRD EDITION
DEITEL® DEVELOPER SERIES

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 U.S., please contact international@pearsoned.com.

Visit us on the web: informit.com/ph

Library of Congress Cataloging-in-Publication Data

On file

© 2016 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. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-13428936-6

ISBN-10: 0-13-428936-6

Text printed in the United States at Edwards Brothers Malloy in Ann Arbor, Michigan.

First printing, November 2015

ANDROID™ 6 FOR PROGRAMMERS

AN APP-DRIVEN APPROACH

THIRD EDITION

DEITEL® DEVELOPER SERIES

Paul Deitel • Harvey Deitel • Alexander Wald
Deitel & Associates, Inc.



Boston • Columbus • Indianapolis • New York • San Francisco
Amsterdam • Capetown • Dubai • London • Madrid • Milan • Munich
Paris • Montreal • Toronto • Delhi • Mexico City • Sao Paulo • Sidney
Hong Kong • Seoul • Singapore • Taipei • Tokyo

Deitel® Series Page

Deitel® Developer Series

Android™ 6 for Programmers: An App-Driven Approach, 3/E
C for Programmers with an Introduction to C11
C++11 for Programmers
C# 2015 for Programmers
iOS® 8 for Programmers: An App-Driven Approach with Swift™
Java™ for Programmers, 3/E
JavaScript for Programmers
Swift™ for Programmers

How To Program Series

Android™ How to Program, 3/E
C++ How to Program, 9/E
C How to Program, 7/E
Java™ How to Program, Early Objects Version, 10/E
Java™ How to Program, Late Objects Version, 10/E
Internet & World Wide Web How to Program, 5/E
Visual Basic® 2015 How to Program, 7/E
Visual C#® 2015 How to Program, 6/E

Simply Series

Simply C++: An App-Driven Tutorial Approach
Simply Java™ Programming: An App-Driven Tutorial Approach

(continued in next column)

(continued from previous column)

Simply C#: An App-Driven Tutorial Approach
Simply Visual Basic® 2010: An App-Driven Approach, 4/E

CourseSmart Web Books

www.deitel.com/books/CourseSmart/

C++ How to Program, 8/E and 9/E
Simply C++: An App-Driven Tutorial Approach
Java™ How to Program, 9/E and 10/E
Simply Visual Basic® 2010: An App-Driven Approach, 4/E
Visual Basic® 2015 How to Program, 6/E
Visual Basic® 2012 How to Program, 5/E
Visual C#® 2015 How to Program, 5/E
Visual C#® 2012 How to Program, 4/E

LiveLessons Video Learning Products

www.deitel.com/books/LiveLessons/

Android™ 6 App Development Fundamentals, 3/e
C++ Fundamentals
Java™ Fundamentals, 2/e
C# 2015 Fundamentals
C# 2012 Fundamentals
iOS® 8 App Development Fundamentals, 3/e
JavaScript Fundamentals
Swift™ Fundamentals

To receive updates on Deitel publications, Resource Centers, training courses, partner offers and more, please join the Deitel communities on

- Facebook®—[facebook.com/DeitelFan](https://www.facebook.com/DeitelFan)
- Twitter®—[@deitel](https://twitter.com/deitel)
- Google+™—[google.com/+DeitelFan](https://plus.google.com/+DeitelFan)
- YouTube™—[youtube.com/DeitelTV](https://www.youtube.com/DeitelTV)
- LinkedIn®—[linkedin.com/company/deitel-&-associates](https://www.linkedin.com/company/deitel-&-associates)

and register for the free *Deitel® Buzz Online* e-mail newsletter at:

www.deitel.com/newsletter/subscribe.html

To communicate with the authors, send e-mail to:

deitel@deitel.com

For information on *Dive-Into® Series* on-site seminars offered by Deitel & Associates, Inc. worldwide, write to us at deitel@deitel.com or visit:

www.deitel.com/training/

For continuing updates on Pearson/Deitel publications visit:

www.deitel.com
www.pearsonhighered.com/deitel/

Visit the Deitel Resource Centers that will help you master programming languages, software development, Android and iOS app development, and Internet- and web-related topics:

www.deitel.com/ResourceCenters.html

To the Android software-engineering community:

*For creating and evolving a platform that challenges
app developers to test the limits of their imagination*

Paul and Harvey Deitel

Trademarks

DEITEL, the double-thumbs-up bug and DIVE-INTO are registered trademarks of Deitel & Associates, Inc. Java is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Google, Android, Google Play, Google Maps, Google Wallet, Nexus, YouTube, AdSense and AdMob are trademarks of Google, Inc.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided “as is” without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screenshots may be viewed in full within the software version specified.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. Screenshots and icons reprinted with permission from the Microsoft Corporation. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

Throughout this book, trademarks are used. Rather than put a trademark symbol in every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner, with no intention of infringement of the trademark.



Contents

Preface

xxi

Before You Begin

xxxix

I	Introduction to Android	I
1.1	Introduction	2
1.2	Android—The World’s Leading Mobile Operating System	3
1.3	Android Features	3
1.4	Android Operating System	6
1.4.1	Android 2.2 (Froyo)	7
1.4.2	Android 2.3 (Gingerbread)	7
1.4.3	Android 3.0 through 3.2 (Honeycomb)	8
1.4.4	Android 4.0 through 4.0.4 (Ice Cream Sandwich)	8
1.4.5	Android 4.1–4.3 (Jelly Bean)	9
1.4.6	Android 4.4 (KitKat)	10
1.4.7	Android 5.0 and 5.1 (Lollipop)	11
1.4.8	Android 6 (Marshmallow)	12
1.5	Downloading Apps from Google Play	13
1.6	Packages	14
1.7	Android Software Development Kit (SDK)	16
1.8	Object-Oriented Programming: A Quick Refresher	18
1.8.1	The Automobile as an Object	19
1.8.2	Methods and Classes	19
1.8.3	Instantiation	19
1.8.4	Reuse	19
1.8.5	Messages and Method Calls	19
1.8.6	Attributes and Instance Variables	20
1.8.7	Encapsulation	20
1.8.8	Inheritance	20
1.8.9	Object-Oriented Analysis and Design (OOAD)	20
1.9	Test-Driving the Tip Calculator App in an Android Virtual Device (AVD)	21
1.9.1	Opening the Tip Calculator App’s Project in Android Studio	22
1.9.2	Creating Android Virtual Devices (AVDs)	24
1.9.3	Running the Tip Calculator App on the Nexus 6 Smartphone AVD	25
1.9.4	Running the Tip Calculator App on an Android Device	30
1.10	Building Great Android Apps	30

1.11	Android Development Resources	32
1.12	Wrap-Up	34

2 Welcome App 35

Dive-Into® Android Studio: Introducing Visual GUI Design, Layouts, Accessibility and Internationalization

2.1	Introduction	36
2.2	Technologies Overview	37
2.2.1	Android Studio	37
2.2.2	LinearLayout, TextView and ImageView	37
2.2.3	Extensible Markup Language (XML)	38
2.2.4	App Resources	38
2.2.5	Accessibility	38
2.2.6	Internationalization	38
2.3	Creating an App	38
2.3.1	Launching Android Studio	39
2.3.2	Creating a New Project	39
2.3.3	Create New Project Dialog	40
2.3.4	Target Android Devices Step	40
2.3.5	Add an Activity to Mobile Step	42
2.3.6	Customize the Activity Step	43
2.4	Android Studio Window	44
2.4.1	Project Window	45
2.4.2	Editor Windows	46
2.4.3	Component Tree Window	46
2.4.4	App Resource Files	46
2.4.5	Layout Editor	47
2.4.6	Default GUI	48
2.4.7	XML for the Default GUI	49
2.5	Building the App's GUI with the Layout Editor	49
2.5.1	Adding an Image to the Project	50
2.5.2	Adding an App Icon	51
2.5.3	Changing RelativeLayout to a LinearLayout	52
2.5.4	Changing the LinearLayout's id and orientation	53
2.5.5	Configuring the TextView's id and text Properties	54
2.5.6	Configuring the TextView's textSize Property—Scaled Pixels and Density-Independent Pixels	56
2.5.7	Setting the TextView's textColor Property	57
2.5.8	Setting the TextView's gravity Property	58
2.5.9	Setting the TextView's layout:gravity Property	59
2.5.10	Setting the TextView's layout:weight Property	60
2.5.11	Adding an ImageView to Display the Image	60
2.5.12	Previewing the Design	64
2.6	Running the Welcome App	65
2.7	Making Your App Accessible	66

2.8	Internationalizing Your App	67
2.8.1	Localization	67
2.8.2	Naming the Folders for Localized Resources	68
2.8.3	Adding String Translations to the App's Project	68
2.8.4	Localizing Strings	68
2.8.5	Testing the App in Spanish on an AVD	69
2.8.6	Testing the App in Spanish on a Device	70
2.8.7	TalkBack and Localization	71
2.8.8	Localization Checklist	71
2.8.9	Professional Translation	71
2.9	Wrap-Up	72

3 Tip Calculator App 73

Introducing GridLayout, EditText, SeekBar, Event Handling, NumberFormat, Customizing the App's Theme and Defining App Functionality with Java

3.1	Introduction	74
3.2	Test-Driving the Tip Calculator App	75
3.3	Technologies Overview	76
3.3.1	Class Activity	76
3.3.2	Activity Lifecycle Methods	77
3.3.3	AppCompat Library and Class AppCompatActivity	77
3.3.4	Arranging Views with a GridLayout	78
3.3.5	Creating and Customizing the GUI with the Layout Editor and the Component Tree and Properties Windows	78
3.3.6	Formatting Numbers as Locale-Specific Currency and Percentage Strings	78
3.3.7	Implementing Interface TextWatcher for Handling EditText Text Changes	79
3.3.8	Implementing Interface OnSeekBarChangeListener for Handling SeekBar Thumb Position Changes	79
3.3.9	Material Themes	79
3.3.10	Material Design: Elevation and Shadows	80
3.3.11	Material Design: Colors	80
3.3.12	AndroidManifest.xml	81
3.3.13	Searching in the Properties Window	81
3.4	Building the GUI	81
3.4.1	GridLayout Introduction	81
3.4.2	Creating the TipCalculator Project	82
3.4.3	Changing to a GridLayout	83
3.4.4	Adding the TextViews, EditText and SeekBar	83
3.4.5	Customizing the Views	86
3.5	Default Theme and Customizing Theme Colors	88
3.5.1	parent Themes	88
3.5.2	Customizing Theme Colors	89
3.5.3	Common View Property Values as Styles	91

3.6	Adding the App's Logic	92
3.6.1	package and import Statements	93
3.6.2	MainActivity Subclass of AppCompatActivity	94
3.6.3	Class Variables and Instance Variables	94
3.6.4	Overriding Activity Method onCreate	95
3.6.5	MainActivity Method calculate	97
3.6.6	Anonymous Inner Class That Implements Interface OnSeekBarChangeListener	98
3.6.7	Anonymous Inner Class That Implements Interface TextWatcher	99
3.7	AndroidManifest.xml	100
3.7.1	manifest Element	101
3.7.2	application Element	101
3.7.3	activity Element	102
3.7.4	intent-filter Element	102
3.8	Wrap-Up	103

4 Flag Quiz App 105

Fragments, Menus, Preferences, Explicit Intents, Handler, AssetManager, Tweened Animations, Animators, Toasts, Color State Lists, Layouts for Multiple Device Orientations, Logging Error Messages for Debugging

4.1	Introduction	106
4.2	Test-Driving the Flag Quiz App	108
4.2.1	Configuring the Quiz's Settings	108
4.2.2	Taking the Quiz	110
4.3	Technologies Overview	113
4.3.1	Menus	113
4.3.2	Fragments	113
4.3.3	Fragment Lifecycle Methods	114
4.3.4	Managing Fragments	114
4.3.5	Preferences	114
4.3.6	assets Folder	115
4.3.7	Resource Folders	115
4.3.8	Supporting Different Screen Sizes and Resolutions	116
4.3.9	Determining the Device Orientation	117
4.3.10	Toasts for Displaying Messages	117
4.3.11	Using a Handler to Execute a Runnable in the Future	117
4.3.12	Applying an Animation to a View	117
4.3.13	Using ViewAnimationUtils to Create a Circular Reveal Animator	118
4.3.14	Specifying Colors Based on a View's State Via a Color State List	118
4.3.15	AlertDialog	118
4.3.16	Logging Exception Messages	119
4.3.17	Launching Another Activity Via an Explicit Intent	119
4.3.18	Java Data Structures	120
4.3.19	Java SE 7 Features	120
4.3.20	AndroidManifest.xml	120

4.4	Creating the Project, Resource Files and Additional Classes	121
4.4.1	Creating the Project	121
4.4.2	Blank Activity Template Layouts	121
4.4.3	Configuring Java SE 7 Support	122
4.4.4	Adding the Flag Images to the Project	122
4.4.5	strings.xml and Formatted String Resources	123
4.4.6	arrays.xml	124
4.4.7	colors.xml	126
4.4.8	button_text_color.xml	126
4.4.9	Editing menu_main.xml	127
4.4.10	Creating the Flag Shake Animation	128
4.4.11	preferences.xml for Specifying the App's Settings	129
4.4.12	Adding Classes <code>SettingsActivity</code> and <code>SettingsActivityFragment</code> to the Project	131
4.5	Building the App's GUI	132
4.5.1	activity_main.xml Layout for Devices in Portrait Orientation	132
4.5.2	Designing fragment_main.xml Layout	132
4.5.3	Graphical Layout Editor Toolbar	138
4.5.4	content_main.xml Layout for Tablet Landscape Orientation	138
4.6	MainActivity Class	140
4.6.1	package Statement and import Statements	140
4.6.2	Fields	141
4.6.3	Overridden Activity Method <code>onCreate</code>	141
4.6.4	Overridden Activity Method <code>onStart</code>	143
4.6.5	Overridden Activity Method <code>onCreateOptionsMenu</code>	144
4.6.6	Overridden Activity Method <code>onOptionsItemSelected</code>	145
4.6.7	Anonymous Inner Class That Implements <code>OnSharedPreferenceChangeListener</code>	145
4.7	MainActivityFragment Class	147
4.7.1	package and import Statements	147
4.7.2	Fields	148
4.7.3	Overridden Fragment Method <code>onCreateView</code>	149
4.7.4	Method <code>updateGuessRows</code>	151
4.7.5	Method <code>updateRegions</code>	152
4.7.6	Method <code>resetQuiz</code>	152
4.7.7	Method <code>loadNextFlag</code>	154
4.7.8	Method <code>getCountryName</code>	156
4.7.9	Method <code>animate</code>	156
4.7.10	Anonymous Inner Class That Implements <code>OnClickListener</code>	158
4.7.11	Method <code>disableButtons</code>	160
4.8	SettingsActivity Class	161
4.9	SettingsActivityFragment Class	161
4.10	AndroidManifest.xml	162
4.11	Wrap-Up	164

5 Doodlz App

165

2D Graphics, Canvas, Bitmap, Accelerometer, SensorManager, Multitouch Events, MediaStore, Printing, Android 6.0 Permissions, Gradle

5.1	Introduction	167
5.2	Test-Driving the Doodlz App in an Android Virtual Device (AVD)	168
5.3	Technologies Overview	173
5.3.1	Activity and Fragment Lifecycle Methods	173
5.3.2	Custom Views	174
5.3.3	Using SensorManager to Listen for Accelerometer Events	174
5.3.4	Custom DialogFragments	174
5.3.5	Drawing with Canvas, Paint and Bitmap	175
5.3.6	Processing Multiple Touch Events and Storing Lines in Paths	175
5.3.7	Saving to the Device	175
5.3.8	Printing and the Android Support Library's PrintHelper Class	176
5.3.9	New Android 6.0 (Marshmallow) Permissions Model	176
5.3.10	Adding Dependencies Using the Gradle Build System	176
5.4	Creating the Project and Resources	176
5.4.1	Creating the Project	176
5.4.2	Gradle: Adding a Support Library to the Project	177
5.4.3	strings.xml	177
5.4.4	Importing the Material Design Icons for the App's Menu Items	178
5.4.5	MainActivityFragment Menu	179
5.4.6	Adding a Permission to AndroidManifest.xml	181
5.5	Building the App's GUI	181
5.5.1	content_main.xml Layout for MainActivity	181
5.5.2	fragment_main.xml Layout for MainActivityFragment	181
5.5.3	fragment_color.xml Layout for ColorDialogFragment	182
5.5.4	fragment_line_width.xml Layout for LineWidthDialogFragment	185
5.5.5	Adding Class EraseImageDialogFragment	187
5.6	MainActivity Class	187
5.7	MainActivityFragment Class	188
5.7.1	package Statement, import Statements and Fields	188
5.7.2	Overridden Fragment Method onCreateView	189
5.7.3	Methods onResume and enableAccelerometerListening	190
5.7.4	Methods onPause and disableAccelerometerListening	191
5.7.5	Anonymous Inner Class for Processing Accelerometer Events	192
5.7.6	Method confirmErase	193
5.7.7	Overridden Fragment Methods onCreateOptionsMenu and onOptionsItemSelected	193
5.7.8	Method saveImage	195
5.7.9	Overridden Method onRequestPermissionsResult	196
5.7.10	Methods getDoodleView and setDialogOnScreen	197
5.8	DoodleView Class	198
5.8.1	package Statement and import Statements	198
5.8.2	static and Instance Variables	198
5.8.3	Constructor	199

5.8.4	Overridden View Method <code>onSizeChanged</code>	200
5.8.5	Methods <code>clear</code> , <code>setDrawingColor</code> , <code>getDrawingColor</code> , <code>setLineWidth</code> and <code>getLineWidth</code>	200
5.8.6	Overridden View Method <code>onDraw</code>	201
5.8.7	Overridden View Method <code>onTouchEvent</code>	202
5.8.8	<code>touchStarted</code> Method	203
5.8.9	<code>touchMoved</code> Method	204
5.8.10	<code>touchEnded</code> Method	205
5.8.11	Method <code>saveImage</code>	205
5.8.12	Method <code>printImage</code>	206
5.9	<code>ColorDialogFragment</code> Class	207
5.9.1	Overridden DialogFragment Method <code>onCreateDialog</code>	208
5.9.2	Method <code>getDoodleFragment</code>	209
5.9.3	Overridden Fragment Lifecycle Methods <code>onAttach</code> and <code>onDetach</code>	209
5.9.4	Anonymous Inner Class That Responds to the Events of the Alpha, Red, Green and Blue SeekBars	210
5.10	<code>LineWidthDialogFragment</code> Class	211
5.10.1	Method <code>onCreateDialog</code>	213
5.10.2	Anonymous Inner Class That Responds to the Events of the <code>widthSeekBar</code>	214
5.11	<code>EraseImageDialogFragment</code> Class	214
5.12	Wrap-Up	216

6 Cannon Game App

217

*Manual Frame-By-Frame Animation, Graphics, Sound, Threading,
SurfaceView and SurfaceHolder, Immersive Mode and Full-Screen*

6.1	Introduction	218
6.2	Test-Driving the Cannon Game App	220
6.3	Technologies Overview	220
6.3.1	Using the Resource Folder <code>res/raw</code>	220
6.3.2	Activity and Fragment Lifecycle Methods	220
6.3.3	Overriding View Method <code>onTouchEvent</code>	220
6.3.4	Adding Sound with <code>SoundPool</code> and <code>AudioManager</code>	220
6.3.5	Frame-by-Frame Animation with Threads, <code>SurfaceView</code> and <code>SurfaceHolder</code>	221
6.3.6	Simple Collision Detection	221
6.3.7	Immersive Mode	222
6.4	Building the GUI and Resource Files	222
6.4.1	Creating the Project	222
6.4.2	Adjusting the Theme to Remove the App Title and App Bar	223
6.4.3	<code>strings.xml</code>	223
6.4.4	Colors	223
6.4.5	Adding the Sounds to the App	223
6.4.6	Adding Class <code>MainActivityFragment</code>	224
6.4.7	Editing <code>activity_main.xml</code>	224
6.4.8	Adding the <code>CannonView</code> to <code>fragment_main.xml</code>	224

6.5	Overview of This App's Classes	225
6.6	MainActivity Subclass of Activity	226
6.7	MainActivityFragment Subclass of Fragment	226
6.8	Class GameElement	228
	6.8.1 Instance Variables and Constructor	229
	6.8.2 Methods update, draw, and playSound	229
6.9	Blocker Subclass of GameElement	230
6.10	Target Subclass of GameElement	230
6.11	Cannon Class	231
	6.11.1 Instance Variables and Constructor	231
	6.11.2 Method align	232
	6.11.3 Method fireCannonball	232
	6.11.4 Method draw	233
	6.11.5 Methods getCannonball and removeCannonball	234
6.12	Cannonball Subclass of GameElement	234
	6.12.1 Instance Variables and Constructor	234
	6.12.2 Methods getRadius, collidesWith, isOnScreen, and reverseVelocityX	235
	6.12.3 Method update	236
	6.12.4 Method draw	236
6.13	CannonView Subclass of SurfaceView	237
	6.13.1 package and import Statements	237
	6.13.2 Instance Variables and Constants	238
	6.13.3 Constructor	239
	6.13.4 Overriding View Method onSizeChanged	241
	6.13.5 Methods getScreenWidth, getScreenHeight, and playSound	241
	6.13.6 Method newGame	242
	6.13.7 Method updatePositions	244
	6.13.8 Method alignAndFireCannonball	245
	6.13.9 Method showGameOverDialog	246
	6.13.10 Method drawGameElements	247
	6.13.11 Method testForCollisions	248
	6.13.12 Methods stopGame and releaseResources	249
	6.13.13 Implementing the SurfaceHolder.Callback Methods	250
	6.13.14 Overriding View Method onTouchEvent	251
	6.13.15 CannonThread: Using a Thread to Create a Game Loop	252
	6.13.16 Methods hideSystemBars and showSystemBars	253
6.14	Wrap-Up	254

7 WeatherViewer App **256**

REST Web Services, AsyncTask, HttpURLConnection, Processing JSON Responses, JSONObject, JSONArray, ListView, ArrayAdapter, ViewHolder Pattern, TextInputLayout, FloatingActionButton

7.1	Introduction	257
-----	--------------	-----

7.2	Test-Driving the WeatherViewer App	258
7.3	Technologies Overview	259
7.3.1	Web Services	259
7.3.2	JavaScript Object Notation (JSON) and the <code>org.json</code> Package	261
7.3.3	<code>URLConnection</code> Invoking a REST Web Service	263
7.3.4	Using <code>AsyncTask</code> to Perform Network Requests Outside the GUI Thread	263
7.3.5	<code>ListView</code> , <code>ArrayAdapter</code> and the View-Holder Pattern	263
7.3.6	<code>FloatingActionButton</code>	264
7.3.7	<code>TextInputLayout</code>	265
7.3.8	<code>Snackbar</code>	265
7.4	Building the App's GUI and Resource Files	265
7.4.1	Creating the Project	265
7.4.2	<code>AndroidManifest.xml</code>	265
7.4.3	<code>strings.xml</code>	266
7.4.4	<code>colors.xml</code>	266
7.4.5	<code>activity_main.xml</code>	266
7.4.6	<code>content_main.xml</code>	267
7.4.7	<code>list_item.xml</code>	268
7.5	Class <code>Weather</code>	269
7.5.1	package Statement, import Statements and Instance Variables	270
7.5.2	Constructor	270
7.5.3	Method <code>convertTimeStampToDay</code>	271
7.6	Class <code>WeatherArrayAdapter</code>	272
7.6.1	package Statement and import Statements	272
7.6.2	Nested Class <code>ViewHolder</code>	273
7.6.3	Instance Variable and Constructor	273
7.6.4	Overridden <code>ArrayAdapter</code> Method <code>getView</code>	273
7.6.5	<code>AsyncTask</code> Subclass for Downloading Images in a Separate Thread	275
7.7	Class <code>MainActivity</code>	277
7.7.1	package Statement and import Statements	277
7.7.2	Instance Variables	278
7.7.3	Overridden Activity Method <code>onCreate</code>	279
7.7.4	Methods <code>dismissKeyboard</code> and <code>createUrl</code>	280
7.7.5	<code>AsyncTask</code> Subclass for Invoking a Web Service	281
7.7.6	Method <code>convertJSONtoArrayList</code>	283
7.8	Wrap-Up	284

8 **Twitter® Searches App** **286**

SharedPreferences, SharedPreferences.Editor, Implicit Intents, Intent Choosers, RecyclerView, RecyclerView.Adapter, RecyclerView.ViewHolder, RecyclerView.ItemDecoration

8.1	Introduction	287
8.2	Test-Driving the App	288
8.2.1	Adding a Favorite Search	288

8.2.2	Viewing Twitter Search Results	291
8.2.3	Editing a Search	291
8.2.4	Sharing a Search	293
8.2.5	Deleting a Search	294
8.2.6	Scrolling Through Saved Searches	295
8.3	Technologies Overview	295
8.3.1	Storing Key–Value Data in a SharedPreferences File	295
8.3.2	Implicit Intents and Intent Choosers	296
8.3.3	RecyclerView	296
8.3.4	RecyclerView.Adapter and RecyclerView.ViewHolder	297
8.3.5	RecyclerView.ItemDecoration	297
8.3.6	Displaying a List of Options in an AlertDialog	297
8.4	Building the App’s GUI and Resource Files	297
8.4.1	Creating the Project	298
8.4.2	AndroidManifest.xml	298
8.4.3	Adding the RecyclerView Library	298
8.4.4	colors.xml	298
8.4.5	strings.xml	299
8.4.6	arrays.xml	299
8.4.7	dimens.xml	299
8.4.8	Adding the Save Button Icon	300
8.4.9	activity_main.xml	300
8.4.10	content_main.xml	300
8.4.11	RecyclerView Item’s Layout: list_item.xml	302
8.5	MainActivity Class	303
8.5.1	package and import Statements	304
8.5.2	MainActivity Fields	304
8.5.3	Overriden Activity Method onCreate	305
8.5.4	TextWatcher Event Handler and Method updateSaveFAB	307
8.5.5	saveButton’s OnClickListener	308
8.5.6	addTaggedSearch Method	309
8.5.7	Anonymous Inner Class That Implements View.OnClickListener to Display Search Results	310
8.5.8	Anonymous Inner Class That Implements View.OnLongClickListener to Share, Edit or Delete a Search	311
8.5.9	shareSearch Method	313
8.5.10	deleteSearch Method	314
8.6	SearchesAdapter Subclass of RecyclerView.Adapter	315
8.6.1	package Statement, import statements, Instance Variables and Constructor	315
8.6.2	Nested ViewHolder Subclass of RecyclerView.ViewHolder	316
8.6.3	Overridden RecyclerView.Adapter Methods	317
8.7	ItemDivider Subclass of RecyclerView.ItemDecoration	318
8.8	A Note on Fabric: Twitter’s New Mobile Development Platform	320
8.9	Wrap-Up	320

9 Address Book App 322

FragmentTransactions and the Fragment Back Stack, SQLite, SQLiteDatabase, SQLiteOpenHelper, ContentProvider, ContentResolver, Loader, LoaderManager, Cursor and GUI Styles

9.1	Introduction	324
9.2	Test-Driving the Address Book App	326
9.2.1	Adding a Contact	326
9.2.2	Viewing a Contact	327
9.2.3	Editing a Contact	327
9.2.4	Deleting a Contact	327
9.3	Technologies Overview	328
9.3.1	Displaying Fragments with <code>FragmentTransactions</code>	328
9.3.2	Communicating Data Between a Fragment and a Host Activity	329
9.3.3	Manipulating a <code>SQLite</code> Database	329
9.3.4	<code>ContentProviders</code> and <code>ContentResolvers</code>	329
9.3.5	<code>Loader</code> and <code>LoaderManager</code> —Asynchronous Database Access	330
9.3.6	Defining Styles and Applying Them to GUI Components	331
9.3.7	Specifying a <code>TextView</code> Background	331
9.4	Building the GUI and Resource Files	331
9.4.1	Creating the Project	331
9.4.2	Creating the App's Classes	331
9.4.3	Add the App's Icons	333
9.4.4	<code>strings.xml</code>	333
9.4.5	<code>styles.xml</code>	334
9.4.6	<code>textview_border.xml</code>	335
9.4.7	<code>MainActivity</code> 's Layout	335
9.4.8	<code>ContactsFragment</code> 's Layout	337
9.4.9	<code>DetailFragment</code> 's Layout	338
9.4.10	<code>AddEditFragment</code> 's Layout	339
9.4.11	<code>DetailFragment</code> 's Menu	341
9.5	Overview of This Chapter's Classes	341
9.6	<code>DatabaseDescription</code> Class	342
9.6.1	static Fields	342
9.6.2	Nested Class <code>Contact</code>	343
9.7	<code>AddressBookDatabaseHelper</code> Class	344
9.8	<code>AddressBookContentProvider</code> Class	346
9.8.1	<code>AddressBookContentProvider</code> Fields	346
9.8.2	Overridden Methods <code>onCreate</code> and <code>getType</code>	347
9.8.3	Overridden Method <code>query</code>	348
9.8.4	Overridden Method <code>insert</code>	350
9.8.5	Overridden Method <code>update</code>	352
9.8.6	Overridden Method <code>delete</code>	353
9.9	<code>MainActivity</code> Class	354
9.9.1	Superclass, Implemented Interfaces and Fields	354
9.9.2	Overridden Method <code>onCreate</code>	355
9.9.3	<code>ContactsFragment.ContactListener</code> Methods	356

9.9.4	Method displayContact	357
9.9.5	Method displayAddEditFragment	358
9.9.6	DetailFragment.DetailFragmentListener Methods	358
9.9.7	AddEditFragment.AddEditFragmentListener Method	359
9.10	ContactsFragment Class	360
9.10.1	Superclass and Implemented Interface	360
9.10.2	ContactsFragmentListener	360
9.10.3	Fields	361
9.10.4	Overridden Fragment Method onCreateView	361
9.10.5	Overridden Fragment Methods onAttach and onDetach	363
9.10.6	Overridden Fragment Method onActivityCreated	363
9.10.7	Method updateContactList	364
9.10.8	LoaderManager.LoaderCallbacks<Cursor> Methods	364
9.11	ContactsAdapter Class	365
9.12	AddEditFragment Class	368
9.12.1	Superclass and Implemented Interface	368
9.12.2	AddEditFragmentListener	369
9.12.3	Fields	369
9.12.4	Overridden Fragment Methods onAttach, onDetach and onCreateView	370
9.12.5	TextWatcher nameChangeListener and Method updateSaveButtonFAB	372
9.12.6	View.OnClickListener saveContactButtonClicked and Method saveContact	373
9.12.7	LoaderManager.LoaderCallbacks<Cursor> Methods	374
9.13	DetailFragment Class	376
9.13.1	Superclass and Implemented Interface	376
9.13.2	DetailFragmentListener	377
9.13.3	Fields	377
9.13.4	Overridden Methods onAttach, onDetach and onCreateView	378
9.13.5	Overridden Methods onCreateOptionsMenu and onOptionsItemSelected	379
9.13.6	Method deleteContact and DialogFragment confirmDelete	380
9.13.7	LoaderManager.LoaderCallback<Cursor> Methods	381
9.14	Wrap-Up	382

10 Google Play and App Business Issues 384

10.1	Introduction	385
10.2	Preparing Your Apps for Publication	385
10.2.1	Testing Your App	386
10.2.2	End User License Agreement	386
10.2.3	Icons and Labels	387
10.2.4	Versioning Your App	387
10.2.5	Licensing to Control Access to Paid Apps	388
10.2.6	Obfuscating Your Code	388

10.2.7	Getting a Private Key for Digitally Signing Your App	388
10.2.8	Featured Image and Screenshots	388
10.2.9	Promotional App Video	390
10.3	Pricing Your App: Free or Fee	390
10.3.1	Paid Apps	391
10.3.2	Free Apps	391
10.4	Monetizing Apps with In-App Advertising	392
10.5	Monetizing Apps: Using In-App Billing to Sell Virtual Goods	393
10.6	Registering at Google Play	394
10.7	Setting Up a Google Payments Merchant Account	395
10.8	Uploading Your Apps to Google Play	395
10.9	Launching Play Store from Within Your App	397
10.10	Managing Your Apps in Google Play	398
10.11	Other Android App Marketplaces	398
10.12	Other Mobile App Platforms and Porting Your Apps	398
10.13	Marketing Your Apps	399
10.14	Wrap-Up	403

Index

This page intentionally left blank



Preface

Welcome to the dynamic world of Android *smartphone* and *tablet* app development with the Android Software Development Kit (SDK), the Java™ programming language and the rapidly evolving Android Studio Integrated Development Environment (IDE). Many of the Android techniques we present also apply to Android Wear and Android TV app development, so after reading this book, you'll be well prepared to investigate developing apps for these platforms.

Android 6 for Programmers: An App-Driven Approach presents leading-edge mobile computing technologies for professional software developers. In our *app-driven approach*, we present concepts in *complete working Android apps*, rather than using code snippets. Chapters 2–9 each present one app. Each chapter begins with an introduction to the app, an app test-drive showing one or more sample executions and an overview of the technologies we used to build the app. Then we present a detailed source-code walkthrough. All of the source code is available at

<http://www.deitel.com/books/AndroidFP3>

We recommend that you view each app's source code in the IDE as you read the chapter.

The opportunities for Android app developers are enormous. Sales of Android devices and app downloads have been growing exponentially. The first-generation Android phones were released in October 2008. According to IDC, after the first three months of 2015, Android had 78% of the global smartphone market share, compared to 18.3% for Apple, 2.7% for Microsoft and 0.3% for Blackberry.¹ Over one billion Android devices shipped in 2014 alone.² At the 2015 Google I/O conference, Google announced that in the prior 12 months there had been 50 billion app installs from Google Play™—Google's marketplace for Android apps.³ Fierce competition among popular mobile platforms and carriers is leading to rapid innovation and falling prices. In addition, competition among the hundreds of Android device manufacturers is driving hardware and software innovation within the Android community.

Copyright Notice and Code License

All of the Android code and Android apps in the book are copyrighted by Deitel & Associates, Inc. The sample Android apps in the book are licensed under a Creative Commons Attribution 3.0 Unported License (<http://creativecommons.org/licenses/by/3.0>), with the exception that they may not be reused in any way in educational tutorials and textbooks, whether in print or digital format. Additionally, the authors and publisher make no warranty of any kind,

1. <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>.
2. <http://www.businessinsider.com/android-1-billion-shipments-2014-strategy-analytics-2015-2>.
3. <http://bit.ly/2015GoogleIOKeynote>.

expressed or implied, with regard to these programs or to the documentation contained in this book. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs. You're welcome to use the apps in the book as shells for your own apps, building on their existing functionality (within the terms of the preceding license). If you have any questions, contact us at deitel@deitel.com.

Intended Audience

We assume that you're a Java programmer with object-oriented programming experience. We also assume that you're familiar with XML—as you'll see, Android projects contain many XML files, though you'll often interact with them through editors that hide much or all of the XML from you. We use only complete, working apps, so if you don't know Java but have object-oriented programming experience in a C-based language such as C++, C#, Swift or Objective-C you should be able to master the material quickly, learning a good amount of Java and Java-style object-oriented programming along the way.

This book is *not* a Java tutorial. If you're interested in learning Java, you may want to check out our publications:

- *Java for Programmers, 3/e* (<http://www.deitel.com/books/javafp3>)
- *Java Fundamentals, 2/e* LiveLessons videos. These videos are available to Safari-BooksOnline.com subscribers and may be purchased from Informit.com and Udemy.com. Visit <http://www.deitel.com/LiveLessons> for subscription and purchase links.
- *Java How to Program, 10/e* (<http://www.deitel.com/books/jhttp10>; ISBN# 0-13-380780-0)

If you're not familiar with XML, many free online tutorials are available, including:

- <http://www.ibm.com/developerworks/xml/newto>
- <http://www.w3schools.com/xml/default.asp>
- <http://bit.ly/DeitelXMLBasics>
- <http://bit.ly/StructureXMLData>

Features

Here are some of this book's key features:

App-Driven Approach. Chapters 2–9 each present one completely coded app—we discuss what the app does, show screenshots of the app in action, test-drive it and overview the technologies and architecture we used to build it. Then we build the app's GUI and resource files, present the complete code and do a detailed code walkthrough. We discuss the programming concepts and demonstrate the functionality of the Android APIs used in the app.

Android 6 SDK. We cover various new Android 6 Software Development Kit (SDK) features.

Android Studio IDE. The free Android Studio (based on IntelliJ IDEA Community Edition) is now Google's preferred IDE for Android app development (the original Android

development tools were based on the Eclipse IDE). Android Studio, combined with the free Android Software Development Kit (SDK) and the free Java Development Kit (JDK), provide all the software you'll need to create, run and debug Android apps, export them for distribution (e.g., upload them to Google Play™) and more. See the Before You Begin section after this Preface for download and installation instructions for all this software.

Material Design. With Android 5, Google introduced its new Android look-and-feel, based on their material design specification:

<http://www.google.com/design/spec/material-design/introduction.html>

In the specification, Google overviews the goals and principles of material design, then provides details on animation techniques, styling on-screen elements, positioning elements, uses of specific user-interface components, user-interaction patterns, accessibility, internationalization and more. Google now uses material-design principles in its mobile and browser-based apps.

Material design is a massive topic. In this book, we focus on the following aspects of material design:

- Using Android's built-in *Material themes*—these give Android's built-in user-interface components a look-and-feel that's consistent with material design principles.
- Using built-in Android Studio *app templates*—these are designed by Google to adhere to material design principles.
- Using *user-interface components*, as appropriate, that are recommended by the material design guidelines for specific purposes, such as `FloatingActionButton`, `TextInputLayout` and `RecyclerView`.

In addition to Google's material design specification, you may want to read the book *Android User Interface Design: Implementing Material Design for Developers, 2nd Edition*:

<http://bit.ly/IanCliftonMaterialDesign>

by our professional colleague and past *Android for Programmers* reviewer Ian Clifton. From Ian: "Google announced the material design guidelines in 2014, creating a design system that suggested how an app should look as well as behave. The goal was to provide a design framework that would improve the visual appearance of all apps and create a behavioral consistency that did not exist previously across apps. *Android User Interface Design: Implementing Material Design for Developers, 2nd Edition* covers material design in detail, making user-centered design, color theory, typography, interaction patterns and other aspects of design accessible to all developers."

Support and App Compatibility Libraries. A big challenge developers face when using new Android features is backward compatibility with earlier Android platforms. Many new Android features are now introduced via support libraries. These enable you to use new features in apps targeting current and past Android platforms. One such library is the `AppCompat` library. Android Studio's app templates have been updated to use the `AppCompat` library and its themes, enabling the new apps you create to run on most Android devices. By creating apps with the `AppCompat` library from the start, you avoid having to reimplement your code if you decide to support older Android versions to target a wider audience.

In addition, at the 2015 Google I/O developer conference, Google introduced the Android Design Support Library

<http://android-developers.blogspot.com/2015/05/android-design-support-library.html>

for using material design in Android 2.1 and higher. Material design support also is built into most of Android Studio's app templates.

REST Web Services and JSON. Chapter 7 presents the **Weather Viewer** app, which demonstrates how to invoke Representational State Transfer (REST) web services—in this case, the 16-day weather-forecast service from `OpenWeatherMap.org`. This web service returns the weather forecast in JavaScript Object Notation (JSON)—a popular text-based data-interchange format used to represent objects as key–value pairs of data. The app also uses classes from the `org.json` package to process the web service's JSON response.

Android 6.0 Permissions. Android 6.0 has a new permissions model that's designed for a better user experience. Before Android 6.0, a user was required at installation time to grant in advance all permissions that an app would ever need, which often discouraged users from installing apps. With the new model, the app is installed without asking for any permissions. Instead, the user is asked to grant a permission only the first time the corresponding feature is used. Chapter 5 introduces the new permissions model and uses it to request permission from the user to store an image on the device's external storage.

Fragments. Starting with Chapter 4, we use Fragments to create and manage portions of each app's GUI. You can combine several fragments to create user interfaces that take advantage of tablet screen sizes. You also can easily interchange fragments to make your GUIs more dynamic, as you'll do in Chapter 9.

View-Holder Pattern, ListView and RecyclerView. The apps in Chapters 7–9 each display scrollable lists of data. Chapter 7 presents the data in a `ListView` and introduces the view-holder pattern, which improves scrolling performance by reusing GUI components that scroll off-screen. With `ListsViews`, using the view-holder pattern is recommended. Chapters 8 and 9 each present a list of data in the more flexible and more efficient `RecyclerView` for which the view-holder pattern is required.

Printing. We demonstrate class `PrintHelper` (Chapter 5) from Android's printing framework for printing from an app. Class `PrintHelper` provides a user interface for selecting a printer, has a method for determining whether a given device supports printing and provides a method for printing a `Bitmap`. `PrintHelper` is part of the Android Support Library.

Immersive Mode. The status bar at the top of the screen and the menu buttons at the bottom can be hidden, allowing your apps to fill more of the screen. Users can access the status bar by swiping down from the top of the screen, and the system bar (with the back button, home button and recent apps button) by swiping up from the bottom.

Testing on Android Smartphones, Tablets and the Android Emulator. For the best app-development experience and results, you should test your apps on actual Android smartphones and tablets. You can still have a meaningful experience using just the Android emulator (see the *Before You Begin* section); however, it's processor intensive and can be slow, particularly with games that have a lot of moving parts. In Chapter 1, we mention some Android features that are not supported on the emulator.

Cloud Test Lab. Google is working on a new *Cloud Test Lab*—an online site for testing your apps across a wide range of devices, device orientations, locales, spoken languages and network conditions. You'll be able to run automated tests and receive detailed reports containing screenshots and videos of your app in action, as well as error logs to help you find problems and improve your apps. For more information and to sign up to be notified when Cloud Test Lab becomes available, visit:

```
http://developers.google.com/cloud-test-lab/
```

Android Wear and Android TV. Android Wear runs on smart watches. Android TV runs directly on some smart TVs and media players that you can connect to your TV (typically via HDMI cables). Many Android techniques we present also apply to Android Wear and Android TV app development. The Android SDK provides Android Wear and Android TV emulators, so you can test your apps for these platforms, even if you don't have devices. To learn more about these technologies from the developer perspective, visit:

```
http://developer.android.com/wear/index.html
```

for Android Wear and

```
http://developer.android.com/tv/index.html
```

for Android TV.

Multimedia. The apps use a range of Android multimedia capabilities, including graphics, images, frame-by-frame animation and audio.

Uploading Apps to Google Play. Chapter 10, Google Play and App Business Issues, discusses Google Play and setting up a merchant account so you can sell your apps. You'll learn how to prepare apps for submission to Google Play, find tips for pricing your apps, and find resources for monetizing them with in-app advertising and in-app sales of virtual goods. You'll also find resources for marketing your apps. Chapter 10 can be read after Chapter 1.

Pedagogic Features

Syntax Coloring. For readability, we syntax color the code, similar to Android Studio's use of syntax coloring. Our syntax-coloring conventions are as follows:

```
comments appear like this
keywords appear like this
constants and literal values appear like this
all other code appears like this
```

Code Highlighting. We emphasize the key code segments in each program by enclosing them in yellow rectangles.

Using Fonts for Emphasis. We use various font conventions:

- The defining occurrences of key terms appear **bold maroon** for easy reference.
- On-screen IDE components appear in **bold Helvetica** (e.g., the **File** menu).
- Program source code appears in *Lucida* (e.g., `int x = 5;`).

In this book you'll create GUIs using a combination of visual programming (point-and-click, drag-and-drop) and writing code. We use different fonts when we refer to GUI elements in program code versus GUI elements displayed in the IDE:

- When we refer to a GUI component that we create in a program, we place its class name and object name in a Lucida font—e.g., `Button saveContactButton`.
- When we refer to a GUI component that's part of the IDE, we place the component's text in a **bold Helvetica** font and use a plain text font for the component's type—e.g., “the **File** menu” or “the **Run** button.”

Using the > Character. We use the > character to indicate selecting a menu item from a menu. For example, we use the notation **File > New** to indicate that you should select the **New** menu item from the **File** menu.

Source Code. All of the book's source code is available for download from

<http://www.deitel.com/books/AndroidFP3>

Documentation. All the Android documentation you'll need to develop Android apps is available at

<http://developer.android.com>

An overview of Android Studio is available at

<http://developer.android.com/tools/studio/index.html>

Chapter Objectives. Each chapter begins with a list of learning objectives.

Figures. Numerous tables, source-code listings and screenshots are included.

Software Engineering. We stress program clarity and performance, and we concentrate on building well-engineered, object-oriented software.

Index. We include an extensive index for reference. The page number of the defining occurrence of each key term is highlighted in **bold maroon**.

Working with Open-Source Apps

The numerous free, open-source Android apps available online are excellent resources for learning Android app development. We encourage you to download open-source apps and read their source code to understand how they work.

Caution: The terms of open-source licenses vary considerably. Some allow you to use the app's source code freely for any purpose, while others stipulate that the code is available for personal use only—not for creating for-sale or publicly available apps. **Be sure to read the licensing agreements carefully. If you wish to create a commercial app based on an open-source app, you should consider having an intellectual-property attorney read the license; be aware that these attorneys charge significant fees.**

Android 6 App-Development Fundamentals LiveLessons Video Training Products

Our *Android 6 App-Development Fundamentals LiveLessons* videos show you what you need to know to start building robust, powerful Android apps with Android 6, the Java™ pro-

programming language and Android Studio. Included are approximately 16–20 hours of expert training synchronized with *Android 6 for Programmers: An App-Driven Approach*. For additional information about Deitel LiveLessons video products, visit

<http://www.deitel.com/livelessons>

or contact us at deitel@deitel.com. You also can access our LiveLessons videos if you have a subscription to SafariBooksOnline.com. For a free 10-day trial, register at

<http://www.safaribooksonline.com/register>

Join the Deitel & Associates, Inc. Social Networking Communities

To receive updates on this and our other publications, new and updated apps, online Resource Centers, instructor-led on-site training courses and more, join the Deitel social networking communities on

- Facebook®—<http://facebook.com/DeitelFan>
- LinkedIn®—<http://bit.ly/DeitelLinkedIn>
- Twitter®—<http://twitter.com/deitel>
- Google+™—<http://google.com/+DeitelFan>
- YouTube®—<http://youtube.com/DeitelTV>

and subscribe to the *Deitel® Buzz Online* newsletter

<http://www.deitel.com/newsletter/subscribe.html>

Contacting the Authors

We'd sincerely appreciate your comments, criticisms, corrections and suggestions for improvement. Please address all questions and other correspondence to

deitel@deitel.com

We'll respond promptly and post corrections and clarifications as Android evolves at:

<http://www.deitel.com/books/AndroidFP3>

and on Facebook, LinkedIn, Twitter, Google+ and the *Deitel® Buzz Online*.

Visit <http://www.deitel.com> to

- download code examples
- check out the growing list of online programming Resource Centers
- receive updates for this book, subscribe to the free *Deitel® Buzz Online* e-mail newsletter at <http://www.deitel.com/newsletter/subscribe.html>
- receive information on our *Dive Into® Series* instructor-led programming-language training courses offered at customer sites worldwide.

Acknowledgments

Thanks to Barbara Deitel for long hours devoted to this project—she created all of our Android Resource Centers and patiently researched hundreds of technical details.

We appreciate the efforts and 20-year mentorship of our friend and professional colleague Mark L. Taub, Editor-in-Chief of the Pearson Technology Group. Mark and his team publish all of our professional books and LiveLessons video products. Michelle Housley recruited distinguished members of the Android community to review the manuscript. We selected the cover art and Chuti Prasertsith designed the cover. John Fuller manages the production of all of our Deitel Developer Series books.

We thank Michael Morgano, a former colleague of ours at Deitel & Associates, Inc., now an Android developer at PHHHOTO, who co-authored the first editions of this book and our book, *iPhone for Programmers: An App-Driven Approach*. Michael is an extraordinarily talented software developer.

Finally, we thank Abbey Deitel, former President of Deitel & Associates, Inc., and a graduate of Carnegie Mellon University's Tepper School of Management where she received a B.S. in Industrial Management. Abbey managed the business operations of Deitel & Associates, Inc. for 17 years, along the way co-authoring a number of our publications, including the previous editions' versions of Chapters 1 and 10.

Reviewers of the Content from Android 6 for Programmers: An App-Driven Approach and Android How to Program Recent Editions

We'd like to thank the following professionals and academics who reviewed this book and/or its previous editions. They scrutinized the text and the code and provided countless suggestions for improving the presentation: Paul Beusterien (Principal, Mobile Developer Solutions), Eric J. Bowden, COO (Safe Driving Systems, LLC), Tony Cantrell (Georgia Northwestern Technical College), Ian G. Clifton (Independent Contractor, Android App Developer and author of *Android User Interface Design: Implementing Material Design for Developers, 2nd Edition*), Daniel Galpin (Android Advocate and author of *Intro to Android Application Development*), Jim Hathaway (Application Developer, Kellogg Company), Douglas Jones (Senior Software Engineer, Fullpower Technologies), Charles Lasky (Nagautuck Community College), Enrique Lopez-Manas (Lead Android Architect, Sixt, and Computer Science Teacher at the University of Alcalá in Madrid), Sebastian Nykopp (Chief Architect, Reaktor), Michael Pardo (Android Developer, Mobiata), Luis Ramirez (Lead Android Engineer at Reverb), Ronan "Zero" Schwarz (CIO, OpenIntents), Arijit Sengupta (Wright State University), Donald Smith (Columbia College), Jesus Ubaldo Quevedo-Torrero (University of Wisconsin, Parkside), Dawn Wick (Southwestern Community College) and Frank Xu (Gannon University).

Well, there you have it! *Android 6 for Programmers: An App-Driven Approach* will quickly get you started developing Android apps with Android 6 and Android Studio. We hope you enjoy reading the book as much as we enjoyed writing it!

Paul Deitel
Harvey Deitel

About the Authors

Paul Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of MIT, where he studied Information Technology. He holds the Java Certified Programmer and Java Certified Developer designations and is an Oracle Java Champion. Paul was also named as a Microsoft® Most Valuable Professional (MVP) for C# in 2012–2014.

Through Deitel & Associates, Inc., he has delivered hundreds of programming courses worldwide to clients, including Cisco, IBM, Siemens, Sun Microsystems, Dell, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey Deitel, are the world's best-selling programming-language textbook/professional book/video authors.

Dr. Harvey Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has over 50 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University—he studied computing in each of these programs before they spun off Computer Science departments. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., in 1991 with his son, Paul. The Deitels' publications have earned international recognition, with translations published in Japanese, German, Russian, Spanish, French, Polish, Italian, Simplified Chinese, Traditional Chinese, Korean, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to corporate, academic, government and military clients.

Alexander Wald, a Deitel summer intern, helped us convert the book and our Android apps from Android 4.3 and 4.4 using Eclipse to Android 6 using Android Studio. Alexander is currently pursuing a B.S. in Computer Science at Worcester Polytechnic Institute with a minor in Electrical Engineering. He became interested in mathematics and the sciences at an early age and has been writing code for approximately 9 years. He's motivated by his passion to be creative and innovative and his interest in sharing his knowledge with others.

About Deitel & Associates, Inc.

Deitel & Associates, Inc., founded by Paul Deitel and Harvey Deitel, is an internationally recognized authoring and corporate training organization, specializing in Android and iOS app development, computer programming languages, object technology and Internet and web software technology. The company's clients include many of the world's largest corporations, government agencies, branches of the military, and academic institutions. The company offers instructor-led training courses delivered at client sites worldwide on major programming languages and platforms, including Android app development, iOS app development, Swift™, Java™, C++, C, Visual C#®, Visual Basic®, Internet and web programming and a growing list of additional programming and software-development courses.

Through its 40-year publishing partnership with Prentice Hall/Pearson, Deitel & Associates, Inc., publishes leading-edge programming professional books, college textbooks and *LiveLessons* video courses. Deitel & Associates, Inc. and the authors can be reached at:

deitel@deitel.com

To learn more about Deitel's *Dive-Into*® Series Corporate Training curriculum, visit

<http://www.deitel.com/training>

To request a proposal for worldwide on-site, instructor-led training at your organization, send an e-mail to deitel@deitel.com.

Individuals wishing to purchase Deitel books and *LiveLessons* video training can do so via links posted at <http://www.deitel.com>. Bulk orders by corporations, the government, the military and academic institutions should be placed directly with Pearson. For more information, visit

<http://www.informit.com/store/sales.aspx>



Before You Begin

In this section, you'll set up your computer for use with this book. Google frequently updates the Android™ development tools, so before reading this section, check the book's website

```
http://www.deitel.com/books/AndroidFP3
```

to see if we've posted an updated version of this Before You Begin section.

Software and Hardware System Requirements

To develop Android apps, you need a Windows®, Linux® or Mac® OS X® system. To view the latest operating-system requirements visit

```
http://developer.android.com/sdk/index.html#Requirements
```

and scroll down to the **System Requirements** heading. We developed the apps in this book using the following software:

- Java SE 7 Software Development Kit
- Android Studio 1.4 Integrated Development Environment (IDE)
- Android 6 SDK (API 23)

You'll see how to obtain each of these in the following sections.

Installing the Java Development Kit (JDK)

Android requires the Java Development Kit version 7 (JDK 7). All Java language features in JDK 7 are supported in Android Studio, but the `try-with-resources` statement is supported only for Android platform versions with API levels 19 and higher. To download JDK 7 for Windows, OS X or Linux, go to

```
http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html
```

Choose the appropriate 32-bit or 64-bit version for your computer hardware and operating system. Be sure to follow the installation instructions at

```
http://docs.oracle.com/javase/7/docs/webnotes/install/index.html
```

Android does not yet support Java 8 language features, such as lambda expressions, new interface features and the stream APIs. You can use JDK 8 (as we did when developing this book's apps), provided that you use no Java 8 language features in your code.

Installing Android Studio

Google's Android Studio comes with the latest Android Software Development Kit (SDK) and is based on the popular Java IDE from JetBrains called IntelliJ® IDEA. To download Android Studio, go to

<http://developer.android.com/sdk/index.html>

and click the **Download Android Studio** button. When the download completes, run the installer and follow the on-screen instructions to complete the installation. If you previously installed an earlier Android Studio version, a **Complete Installation** window will appear at the end of the install process and give you the option to import your previous settings. At the time of this writing, Android Studio 1.4 is the current released version and Android Studio 1.5 is available as an early access release.

Using Early Access Releases

When building apps for release to Google Play or other app stores, it's best to use the currently released version of Android Studio. If you'd like to work with new features in Android Studio early access and beta releases, Google releases these versions in the so-called **Canary Channel** and **Beta Channel**. You can configure Android Studio to obtain updates from these channels. To update Android Studio to the latest early access or beta release:

1. Open Android Studio.
2. In the **Welcome to Android Studio** window, click **Configure**.
3. Click **Check for Update**.
4. In the **Platform and Plugin Updates** dialog, click the **Updates** link.
5. In the **Updates** dialog, select **Canary Channel** or **Beta Channel** from the dropdown to the right of the **Automatically check updates for** checkbox.
6. Click **OK**, then click **Close**.
7. Click **Check for Update** again.
8. The IDE will check for updates and tell you whether there are updates to apply.
9. Click **Update and Restart** to install the latest Android Studio version.

If you've previously opened a project in Android Studio and did not close the project, the IDE skips the **Welcome to Android Studio** window and opens the last project. In this case, you can access the **Updates** dialog on a Mac via **Android Studio > Check for Updates...** or on Windows/Linux via **Help > Check for Update....** Then continue from *Step 4* above. For a Google's list of Android Studio Tips and Tricks, visit:

<http://developer.android.com/sdk/installing/studio-tips.html>

Configure Android Studio to Show Line Numbers

By default, Android Studio does not show line numbers next to the code that you write. To turn on line numbers to make it easier to follow our line-numbered code examples:

1. Open Android Studio ().

2. When the **Welcome to Android Studio** window appears, click **Configure**, then click **Settings** to open the **Default Settings** window. If the **Welcome to Android Studio** window does not appear, use the menus on Mac to select **Android Studio > Preferences...** or on Windows/Linux to select **File > Other Settings > Default Settings...**
3. Expand the **Editor > General** node and select **Appearance**, then ensure that **Show line numbers** is selected and click **OK**.

Configure Android Studio to Disallow Code Folding

By default, Android Studio's code-folding feature is enabled. This feature collapses multiple lines of code into a single line so you can focus on other aspects of the code. For example, all the `import` statements in a Java source-code file can be collapsed into a single line to hide them, or an entire method can be collapsed into a single line. You can expand these lines if you need to look at the code in detail. We disabled this feature in our IDE. If you wish to do so, follow the steps in the preceding section, then under **Editor > General > Code Folding** uncheck **Show code folding outline**.

Android 6 SDK

This book's code examples were written using Android 6. At the time of this writing, the Android 6 SDK was bundled with Android Studio. As new Android versions are released, the latest version will be bundled, which may prevent our apps from compiling properly. When you work with this book, we recommend using Android 6. You can install prior Android platform versions as follows:

1. Open Android Studio ()
2. When the **Welcome to Android Studio** window appears, click **Configure**, then click **SDK Manager** to display the **Android SDK** manager. If a project window appears rather than the **Welcome to Android Studio** window, you can access the **Android SDK** manager via **Tools > Android > SDK Manager**.
3. In the **SDK Platforms** tab, check the versions of Android you wish to install, then click **Apply** and **OK**. The IDE then downloads and installs the additional platform versions. The IDE also will help you keep your installed versions up-to-date.

Creating Android Virtual Devices (AVDs)

The Android SDK's **Android emulator** allows you to test apps on your computer rather than on an Android device—this is essential, of course, if you do not have Android devices. To do so, you create **Android Virtual Devices (AVDs)** that run in the emulator. The emulator can be slow, so most Android developers prefer testing on actual devices. Also, the emulator does not support various features, including phone calls, USB connections, headphones and Bluetooth. For the latest emulator capabilities and limitations, visit

<http://developer.android.com/tools/devices/emulator.html>

That page's **Using Hardware Acceleration** section discusses features that can improve emulator performance, such as using the computer's graphics processing unit (GPU) to in-

crease graphics performance, and using the Intel HAXM (hardware accelerated execution manager) emulator to increase overall AVD performance. There are also faster third-party emulators, such as Genymotion.

After you've installed the Android Studio and before you run an app in the emulator, you must create at least one **Android Virtual Device (AVD)** for Android 6. Each AVD defines the characteristics of the device you wish to emulate, including

- its screen size in pixels
- its pixel density
- its screen's physical size
- the size of the SD card for data storage
- and more.

To test your apps for multiple Android devices, you can create AVDs that emulate each unique device. You also can use Google's new Cloud Test Lab

```
https://developers.google.com/cloud-test-lab/
```

a website that will enable you to upload your app and test it on many of today's popular Android devices. By default, Android Studio creates for you one AVD that's configured to use the version of Android bundled with the IDE. For this book, we use AVDs for two of Google's Android reference devices—the Nexus 6 phone and the Nexus 9 tablet—which run standard Android without the modifications made by many device manufacturers. It's easiest to create AVDs in Android Studio once you already have a project open in the IDE. For this reason, we'll show how to create the Android 6 AVDs in Section 1.9.

Setting Up an Android Device for Testing Apps

Testing apps on Android devices tends to be quicker than using AVDs. In addition, recall that there are some features you can test only on actual devices. To execute your apps on Android devices, follow the instructions at

```
http://developer.android.com/tools/device.html
```

If you're developing on Microsoft Windows, you'll also need the Windows USB driver for Android devices that you installed earlier in this Before You Begin section. In some cases on Windows, you may also need the manufacturer's device-specific USB drivers. For a list of USB driver sites for various device brands, visit

```
http://developer.android.com/tools/extras/oem-usb.html
```

Downloading the Book's Code Examples

The source code for *Android 6 for Programmers: An App-Driven Approach* is available for download at

```
http://www.deitel.com/books/AndroidFP3/
```

Click the **Download Code Examples** link to download a ZIP archive file containing the examples to your computer. Depending on your operating system, double click the ZIP file

to unzip the archive or right click and select the option to extract the archive's contents. Remember where the extracted files are located on your system so you can access them later.

A Note Regarding Android Studio and the Android SDK

If you import one of our apps into Android Studio and it does not compile, this could be the result of updates to Android Studio or the Android platform tools. For such issues, please check Android questions and answers on StackOverflow at:

<http://stackoverflow.com/questions/tagged/android>

and the Google+ Android Development community at:

<http://bit.ly/GoogleAndroidDevelopment>

or write to us at

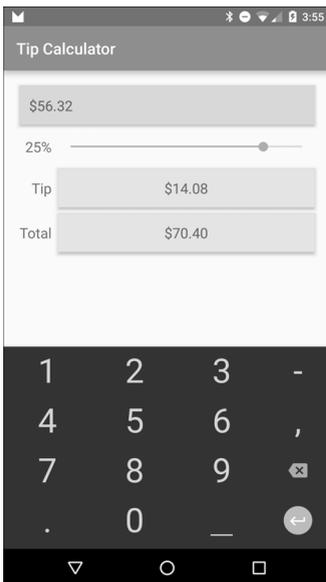
deitel@deitel.com

You've now installed all the software and downloaded the code examples you'll need to study Android app development with *Android 6 for Programmers: An App-Driven Approach* and to begin developing your own apps. Enjoy!

This page intentionally left blank

Tip Calculator App

Introducing GridLayout, EditText, SeekBar, Event Handling, NumberFormat, Customizing the App's Theme and Defining App Functionality with Java



Objectives

In this chapter you'll:

- Change the default GUI theme.
- Customize the GUI theme's colors.
- Design a GUI using a `GridLayout`.
- Use the IDE's **Component Tree** window to add views to a `GridLayout`.
- Use `TextViews`, an `EditText` and a `SeekBar`.
- Use Java object-oriented programming capabilities, including classes, objects, interfaces, anonymous inner classes and inheritance to add functionality to an app.
- Programmatically change the text in a `TextView`.
- Use event handling to respond to user interactions with an `EditText` and a `SeekBar`.
- Specify that the keypad should display by default when the app executes.
- Specify that the app supports only portrait orientation.

3.1 Introduction	3.4.3 Changing to a GridLayout
3.2 Test-Driving the Tip Calculator App	3.4.4 Adding the TextViews, EditText and SeekBar
3.3 Technologies Overview	3.4.5 Customizing the Views
3.3.1 Class Activity	3.5 Default Theme and Customizing Theme Colors
3.3.2 Activity Lifecycle Methods	3.5.1 parent Themes
3.3.3 AppCompatActivity Library and Class AppCompatActivity	3.5.2 Customizing Theme Colors
3.3.4 Arranging Views with a GridLayout	3.5.3 Common View Property Values as Styles
3.3.5 Creating and Customizing the GUI with the Layout Editor and the Component Tree and Properties Windows	3.6 Adding the App's Logic
3.3.6 Formatting Numbers as Locale-Specific Currency and Percentage Strings	3.6.1 package and import Statements
3.3.7 Implementing Interface TextWatcher for Handling EditText Text Changes	3.6.2 MainActivity Subclass of AppCompatActivity
3.3.8 Implementing Interface OnSeekBarChangeListener for Handling SeekBar Thumb Position Changes	3.6.3 Class Variables and Instance Variables
3.3.9 Material Themes	3.6.4 Overriding Activity Method onCreate
3.3.10 Material Design: Elevation and Shadows	3.6.5 MainActivity Method calculate
3.3.11 Material Design: Colors	3.6.6 Anonymous Inner Class That Implements Interface OnSeekBarChangeListener
3.3.12 AndroidManifest.xml	3.6.7 Anonymous Inner Class That Implements Interface TextWatcher
3.3.13 Searching in the Properties Window	3.7 AndroidManifest.xml
3.4 Building the GUI	3.7.1 manifest Element
3.4.1 GridLayout Introduction	3.7.2 application Element
3.4.2 Creating the TipCalculator Project	3.7.3 activity Element
	3.7.4 intent-filter Element
	3.8 Wrap-Up

3.1 Introduction

The **Tip Calculator** app (Fig. 3.1(a)) calculates and displays the tip and total for a restaurant bill amount. As you touch the numeric keypad to enter the bill amount's digits, the app calculates and displays the tip and total bill amounts for the current tip percentage (15% by default). You specify a tip percentage from 0% to 30% by moving the *SeekBar thumb*—this updates the displayed tip percentage and recalculates the tip and total. All numeric values are displayed using *locale-specific* formatting. Figure 3.1(b) shows the app after the user enters the amount 56.32 and changes the tip percentage to 25%.

You'll begin by test-driving the app. Then we'll overview the technologies you'll use to create the app. You'll build the app's GUI using Android Studio's layout editor and the **Component Tree** window. Finally, we'll present the complete Java code for the app and do a detailed code walkthrough.

Note Regarding the Keyboard in Our Screen Captures

The keypad in Fig. 3.1 may differ, based on your AVD's or device's Android version or whether you've installed and selected a custom keyboard on your device. We configured our AVD to display the *dark keyboard* for better contrast in our screen captures. To do so:

1. Touch the home (🏠) icon on your AVD or device.

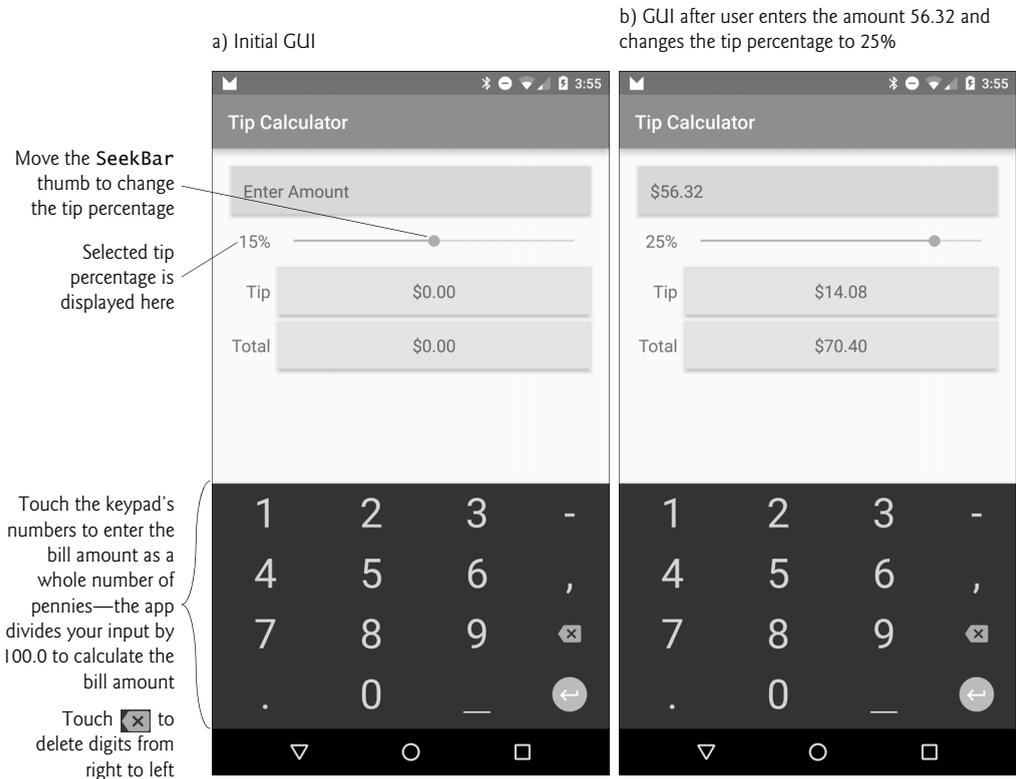


Fig. 3.1 | Entering the bill total and calculating the tip.

2. On the home screen, touch the launcher () icon, then open the **Settings** app.
3. In the **Personal** section, touch **Language and Input**.
4. On an AVD, touch **Android Keyboard (AOSP)**. On a device touch **Google Keyboard**—we assume you're using the standard Android keyboard.
5. Touch **Appearance & layouts**, then touch **Theme**.
6. Touch **Material Dark** to change to the keyboard with the dark background.

3.2 Test-Driving the Tip Calculator App

Opening and Running the App

Perform the steps in Sections 1.9.1 and 1.9.3 to open the **Tip Calculator** app project in Android Studio and run the app on the Nexus 6 AVD. If you prefer, perform the steps in Section 1.9.4 to run the app on an Android phone.

Entering a Bill Total

Enter the bill total 56.32 by touching numbers on the numeric keypad. If you make a mistake, press the keypad's delete button () to erase the last digit you entered. Even though the keypad contains a decimal point, the app is configured so that you may enter only the

digits 0 through 9—other input buttons on the keypad are ignored and an Android device will vibrate to indicate when you touch an invalid input button. Each time you touch a digit or delete a digit, the app reads what you’ve entered so far, and

- converts it to a number
- divides the number by 100.0 and displays the new bill amount
- recalculates the tip and total amounts, based on the current tip percentage (15% by default) and
- displays in the **Tip** and **Total** TextViews the new tip and total amounts.

If you delete all the digits, the app redisplay **Enter Amount** in the blue TextView and displays 0.00 in the orange TextViews. The app divides the value by 100.0 and displays the result in the blue TextView. The app then calculates and updates the tip and total amounts in the orange TextViews.

All monetary amounts are displayed in locale-specific currency formats and the tip percentage is displayed in a locale-specific percentage format. For the U.S. locale, as you enter the four digits 5, 6, 3 and 2, the bill total is displayed successively as **\$0.05**, **\$0.56**, **\$5.63** and **\$56.32**, respectively.

Selecting a Tip Percentage

Use the Seekbar—often called a *slider* in other GUI technologies—to specify the tip percentage. Drag the Seekbar’s *thumb* until the percentage reads 25% (Fig. 3.1(b)). As you drag the thumb, the tip and total update continuously. By default, the Seekbar allows you to select values from 0 to 100, but we specified a maximum value of 30 for this app.

3.3 Technologies Overview

This section introduces the IDE and Android features you’ll use to build the **Tip Calculator** app. We assume that you’re already familiar with Java object-oriented programming—we present Java in our book *Java SE 8 for Programmers* (<http://bit.ly/JavaSE8FP>). You’ll

- use various Android classes to create objects
- call methods on classes and objects
- define and call your own methods
- use inheritance to create a class that defines the **Tip Calculator**’s functionality and
- use event handling, anonymous inner classes and interfaces to process the user’s GUI interactions.

3.3.1 Class Activity

Android apps have four types of executable components—*activities*, *services*, *content providers* and *broadcast receivers*. In this chapter, we’ll discuss activities, which are defined as subclasses of **Activity** (package `android.app`). An app can have many activities, one of which is the first you see after launching the app. You interact with an Activity through views—GUI components that inherit from class **View** (package `android.view`).

Before Android 3.0, a separate Activity was typically associated with each screen of an app. As you’ll see, starting in Chapter 4, an Activity can manage multiple Fragments.

On a phone, each `Fragment` typically occupies the entire screen and the `Activity` switches between the `Fragments`, based on user interactions. On a tablet, activities typically display multiple `Fragments` per screen to take advantage of the larger screen size.

3.3.2 Activity Lifecycle Methods

Throughout its life, an `Activity` can be in one of several *states*—*active* (i.e., *running*), *paused* or *stopped*. The `Activity` transitions between these states in response to various *events*:

- An *active* `Activity` is *visible* on the screen and “has the focus”—that is, it’s in the *foreground*. You can interact with the `Activity` currently in the foreground.
- A *paused* `Activity` is *visible* on the screen but *does not* have the focus—such as when an alert dialog is displayed. You cannot interact with the *paused* activity until it becomes active—for example, after the user dismisses an alert dialog.
- A *stopped* activity is *not visible* on the screen—it’s in the *background* and is likely to be killed by the system when its memory is needed. An `Activity` is *stopped* when another `Activity` enters the *foreground* and becomes *active*. For example, when you answer a phone call, the phone app becomes *active* and the app you previously were using is *stopped*.

As an `Activity` transitions among these states, the Android runtime calls various `Activity lifecycle methods`—all of which are defined by the `Activity` class in package `android.app`. You’ll override the `onCreate` method in *every* activity. This method is called by the Android runtime when an `Activity` is *starting*—that is, when its GUI is about to be displayed so you can interact with the `Activity`. Other lifecycle methods include `onStart`, `onPause`, `onRestart`, `onResume`, `onStop` and `onDestroy`. We’ll discuss most of these in later chapters. Each activity lifecycle method you override *must* call the superclass’s version; otherwise, an *exception* will occur. This is required because each lifecycle method in superclass `Activity` contains code that must execute in addition to the code you define in your overridden lifecycle methods. For more on the `Activity` lifecycle see

<http://developer.android.com/reference/android/app/Activity.html>

3.3.3 AppCompatActivity Library and Class AppCompatActivity

A big challenge developers face when using new Android features is backward compatibility with earlier Android platforms. Google now introduces many new Android features via the *Android Support Library*—a set of libraries that enable you to use newer Android features in apps targeting current and past Android platforms.

One such library is the `AppCompatActivity` library, which enables apps to provide an app bar (formerly called an action bar) and more on devices running Android 2.1 (API 7) and higher—app bars were originally introduced in Android 3.0 (API 11). Android Studio’s app templates have been updated to use the `AppCompatActivity` library, enabling the new apps you create to run on almost all Android devices.

Android Studio’s `Empty Activity` app template defines the app’s `MainActivity` class as a subclass of `AppCompatActivity` (package `android.support.v7.app`)—an indirect subclass of `Activity` that supports using newer Android features in apps running on current and older Android platforms.



Software Engineering Observation 3.1

By creating apps with the `AppCompat` library from the start, you avoid having to reimplement your code if you decide to support older Android versions to target a wider potential audience for your app.



Software Engineering Observation 3.2

Some Android features are not available in earlier Android versions, even if you use the `AppCompat` libraries. For example, Android's printing capabilities are available only in Android 4.4 and higher. If you use such features in your app, you must either restrict the app to the supported platforms or disable those features on Android versions that do not support them.

For more details on Android Support Libraries, including when to use them and how to set them up, visit:

<http://developer.android.com/tools/support-library>

3.3.4 Arranging Views with a `GridLayout`

Recall that you arrange a GUI's views in layouts. We'll use a `GridLayout` (package `android.widget`) to arrange views into cells in a rectangular grid. Cells can occupy *multiple* rows and columns, allowing for complex layouts. Normally, `GridLayout` requires API level 14 or higher. However, the *Android Support Library* provides alternate versions of `GridLayout` and many other views and layouts so that you can use them in older Android versions. For more information on this library and how to use it in your apps, visit

<http://developer.android.com/tools/support-library/index.html>

We'll cover more layouts and views in later chapters—for a complete list, visit

<http://developer.android.com/reference/android/widget/package-summary.html>

3.3.5 Creating and Customizing the GUI with the Layout Editor and the Component Tree and Properties Windows

You'll create `TextViews`, an `EditText` and a `SeekBar` using the layout editor (that you used in Chapter 2) and **Component Tree** window, then customize them with the IDE's **Properties** window.

An `EditText`—often called a *text box* or *text field* in other GUI technologies—is a *subclass* of `TextView` (presented in Chapter 2) that can display text *and* accept text input from the user. You'll specify an `EditText` for numeric input, allow users to enter only digits and restrict the maximum number of digits that can be entered.

A `SeekBar` represents an integer in the range 0–100 by default and allows the user to select a number in that range by moving the `SeekBar`'s thumb. You'll customize the `SeekBar` so the user can choose a tip percentage from the more limited range 0 to 30.

3.3.6 Formatting Numbers as Locale-Specific Currency and Percentage Strings

You'll use class `NumberFormat` (package `java.text`) to create *locale-specific* currency and percentage strings—an important part of *internationalizing* your apps. You also can add

accessibility strings and internationalize the app’s other text using the techniques you learned in Sections 2.7–2.8.

3.3.7 Implementing Interface `TextWatcher` for Handling `EditText` Text Changes

To respond to events when the user changes the text in this app’s `EditText`, you’ll use an *anonymous inner class* to implement the `TextWatcher` interface (from package `android.text`). In particular, you’ll use method `onTextChanged` to display the currency-formatted bill amount and to calculate the tip and total as the user enters each digit. If you’re not familiar with *anonymous inner classes*, visit

<http://bit.ly/AnonymousInnerClasses>

3.3.8 Implementing Interface `SeekBarChangeListener` for Handling `SeekBar` Thumb Position Changes

You’ll use another anonymous inner class to implement the `SeekBar.OnSeekBarChangeListener` interface (from package `android.widget`) to respond to the user moving the `SeekBar`’s *thumb*. In particular, you’ll use method `onProgressChanged` to display the selected tip percentage and to calculate the tip and total as the user moves the `SeekBar`’s thumb.

3.3.9 Material Themes

A theme gives an app a look-and-feel that’s consistent with Android. Projects that you create for Android 5 and higher use themes that adhere to Google’s material design guidelines. There are several predefined material design themes:

- The “light” theme has a white app bar, a white app background and text that is black or shades of dark gray.
- The “light” theme with a dark app bar is the same as above, but the app bar is black with white text by default.
- The “dark” has a black app bar, a dark gray app background and text that is white or shades of light gray.

For each of these themes, there is

- a `Theme.Material` version (e.g., `Theme.Material.Light`) for apps that do not use any `AppCompatActivity` libraries and run on Android 5 and higher, and
- a `Theme.AppCompat` version (e.g., `Theme.AppCompat.Light`) for apps that use `AppCompatActivity` libraries and run on Android 2.1 and higher.

When designing a GUI, you can choose from the predefined themes, or even create your own new ones. For this chapter, we’ll use `Theme.AppCompat.Light.DarkActionBar`, which is the default theme in Android Studio’s app templates. Apps that use the `AppCompatActivity` libraries must use one of the `AppCompatActivity` themes; otherwise, some views will not render correctly. For more information about each theme and to see sample screen captures, visit

<http://www.google.com/design/spec/style/color.html#color-themes>
<http://developer.android.com/training/material/theme.html>



Performance Tip 3.1

Many of today's Android phones use AMOLED displays. On such displays, a black pixel is turned off and does not consume power. Apps that use mostly black themes can reduce power consumption by approximately 40% (<http://bit.ly/AndroidAMOLEDDisplay>).

3.3.10 Material Design: Elevation and Shadows

Google's material design guidelines recommend that objects in your user interfaces cast shadows just as real-world objects do. When you set a view's `elevation` property, Android automatically casts a shadow for that view. Larger `elevation` values result in more pronounced shadows. For this app, we'll set the `elevation` of the blue and orange `TextView`s that display monetary amounts.

The material design guidelines contain elevation recommendations for various on-screen components—for example, a dialog's recommended elevation is 24dp and a menu's is 8dp. For other recommended elevations, see:

<http://www.google.com/design/spec/what-is-material/elevation-shadows.html>

3.3.11 Material Design: Colors

App developers often customize a theme's colors to match a company's branding. If you need to customize theme colors, Google's material design guidelines for color¹ recommend that you choose a color palette consisting of a primary color—with no more than three hues (shades)—and an accent color. The primary colors typically are used to color the status bar and the app bar at the top of the screen and also can be used in your GUI. The accent color is used to tint various views in your GUI, such as `SeekBar`s, `CheckBox`s and `RadioButton`s. Once you choose a palette, you can use Android Studio's **Theme Editor** (Section 3.5.2) to modify a theme's colors.

You can find recommended sample color swatches from the material design color palette at

<http://www.google.com/design/spec/style/color.html#color-color-palette>

For palette color recommendations, visit

<http://www.materialpalette.com/>

This site enables you to click two colors from Google's material design color palette, then it recommends three shades of the primary color, one secondary color and colors for your app's text and icons.

In this app, we'll use color swatches displayed in the Android Studio **Theme Editor** to select

- a blue *primary color* for app bar's background color
- a darker blue *dark primary color* for the status bar that appears above the app bar, and
- an orange *accent color* used to tint the `SeekBar`.

1. <http://www.google.com/design/spec/style/color.html>.

For the amount `TextView`'s light blue color and the tip and total `TextViews`' light orange color, we used Google's material design color palette to choose lighter shades of the primary and accent colors.

3.3.12 `AndroidManifest.xml`

The `AndroidManifest.xml` file is created by the IDE when you create a new app project. This file contains many of the settings that you specify in the **Create New Project** dialog—the app's name, package name and `Activity` name(s) and more. You'll edit this file's XML to add a new setting that forces the *soft keyboard* to be displayed when the app begins executing. You'll also specify that the app supports only *portrait orientation*—that is, the device's longer dimension is vertical.

3.3.13 Searching in the Properties Window

The **Properties** window allows you to search for properties by their names or portions of their names, which can help you find and set properties faster. To do so, click the **Properties** window's title bar and begin typing. At the top of the property list, a **Search for** tooltip appears showing what you've typed so far, and Android Studio highlights parts of every property name in the list that matches all or part of what you've typed. Then you can scroll through the list looking at the property names containing highlights.

The window will also scroll to the specific property that best matches what you type. For example, when searching a `TextView`'s properties, if you type "text co" or "textco", the **Properties** window will highlight portions of many properties, but it specifically scrolls to and highlights the `textColor` property.

3.4 Building the GUI

In this section, we'll show the precise steps for building the **Tip Calculator**'s GUI, including how to customize the Material theme's primary and accent colors.

3.4.1 `GridLayout` Introduction

This app uses a `GridLayout` (package `android.widget`) to arrange views into four *rows* and two *columns*, each indexed from 0 like the elements in an array. You can specify a `GridLayout`'s number of rows and columns in the **Properties** window. Each cell can be *empty* or can hold one or more *views*, including layouts *containing* other views. A row's *height* is determined by the row's *tallest* view. Similarly, a column's *width* is defined by the column's *widest* view. Figure 3.2 shows the **Tip Calculator**'s `GridLayout` labeled by its rows and columns—we drew horizontal lines to delineate the rows and a vertical line to delineate the columns. Views can span *multiple* rows and/or columns—for example, the **Enter Amount** `TextView` in Fig. 3.2 spans both columns in row 0.

When you drag a view onto a `GridLayout` in the **Component Tree**, the view occupies the next available grid cell—cells populate the `GridLayout` left-to-right until a given row is full, then the next view appears in the first column of the next row. As you'll see, you also can specify the exact row and column in which to place a view. We'll discuss other `GridLayout` features as we present the GUI-building steps.

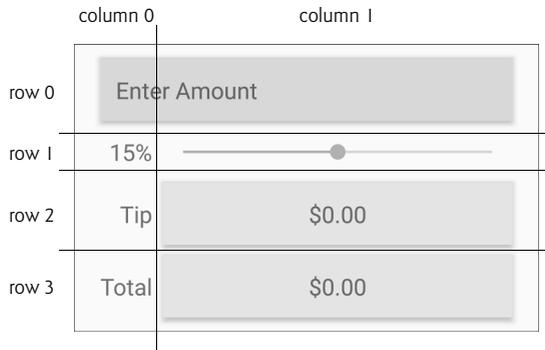


Fig. 3.2 | Tip Calculator GUI’s GridLayout labeled by its rows and columns.

id Property Values for This App’s Views

Figure 3.3 shows the views’ id property values. For clarity, our naming convention is to use the view’s class name in the id property and the corresponding Java variable name. In the first row, there are actually *two* components in the *same* grid cell—the amountTextView (which initially displays **Enter Amount**) *hides* the amountEditText that receives the user input. As you’ll soon see, we restrict the user’s input to whole-number values entered as integer digits, so the user enters the bill amount \$34.56 as 3456. This ensures the user *cannot* enter invalid input. However, this amount should be displayed as *currency*. As the user enters each digit, we divide the amount by 100.0 and display in the amountTextView the locale-specific, currency-formatted amount.

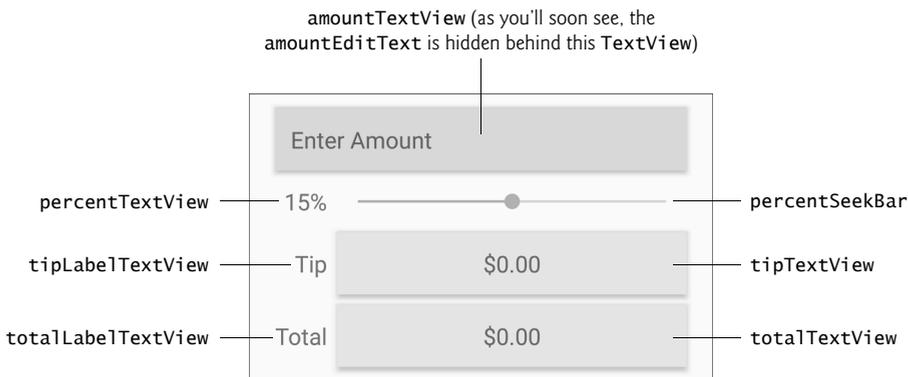


Fig. 3.3 | Tip Calculator views labeled with their id property values.

3.4.2 Creating the TipCalculator Project

Follow the steps in Section 2.3 to create a new project using the **Empty Activity** template. Specify the following values in the **Create New Project** dialog’s **New Project** step:

- **Application name:** Tip Calculator
- **Company Domain:** deitel.com (or specify your own domain name)

For the remaining steps in the **Create New Project** dialog, use the same settings as in Section 2.3. Also, follow the steps in Section 2.5.2 to add an app icon to your project.

Once the project is open in Android Studio, in the layout editor, select **Nexus 6** from the virtual-device drop-down list (Fig. 2.11). Once again, we'll use this device as the basis for our design. Also, delete the **Hello world!** `TextView`.

3.4.3 Changing to a `GridLayout`

Recall that the default layout for an **Empty Activity** is a `RelativeLayout`. Here, you'll change that to a `GridLayout`:

1. Click the **Text** tab at the bottom of the layout editor to switch from the **Design** view to the layout's XML text.
2. At the top of the XML, change `RelativeLayout` to `GridLayout`.
3. Switch back to the layout editor's **Design** tab.

Specifying Two Columns and Default Margins for the `GridLayout`

Recall that the GUI in Fig. 3.2 consists of two columns. To specify this, select `GridLayout` in the **Component Tree** window, then change its `columnCount` property to 2—this property appears near the top of the **Properties** window with the other layout properties. You do not need to set the `rowCount`—it will be increased as we build the GUI.

By default, there are *no margins*—spacing that separates views—around a `GridLayout`'s cells. The material design guidelines recommend 8dp minimum spacing between views:

<http://developer.android.com/design/style/metrics-grids.html>.

`GridLayout` can enforce this recommended spacing. With the `GridLayout` selected in the **Component Tree**, in the **Properties** window, check the `GridLayout`'s `useDefaultMargins` property (which sets it to `true`) to use the recommended margins around the layout's cells.

3.4.4 Adding the `TextViews`, `EditText` and `SeekBar`

You'll now build the GUI in Fig. 3.2. You'll start with the basic layout and views in this section. In Section 3.4.5, you'll customize the views' properties to complete the design. Then, in Section 3.5, you'll change the default theme and customize two of its colors. As you add each view to the GUI, immediately set its `id` property using the names in Fig. 3.3. You'll add views to the `GridLayout` using the **Component Tree** window. If you drop a view in the wrong location in the **Component Tree**, you can drag it to the correct location.

You may also drag views directly onto the layout editor. For a `GridLayout`, the layout editor displays a grid of green guidelines to help you position the view. As you drag a view over the grid, the layout editor displays a tooltip indicating the row and column in which the view will be placed if you drop the view at that location.



Error-Prevention Tip 3.1

The cells in the layout editor's grid of green guidelines are small. If you drop a view in the wrong location, the layout editor might change the `GridLayout`'s `rowCount` and `columnCount` property values and incorrectly set the view's `layout:row` and `layout:column` property values, causing your GUI to lay out incorrectly. If so, reset the `GridLayout`'s `rowCount` and `columnCount`, based on your design, and change the view's `layout:row` and `layout:column` property values to the correct row and column for your design.

Step 1: Adding Views to the First Row

The first row consists of the `amountTextView` and the `amountEditText`—both occupy the same cell and span two columns. Each time you drop a view onto the `GridLayout` in the **Component Tree** window, the view is placed in the layout's *next open cell*, unless you specify otherwise by setting the view's `layout:row` and `layout:column` properties. You'll do that in this step so that the `amountEditText` and `amountTextView` appear in the same cell with the `amountTextView` in the foreground.

This app's `TextView`s use the *medium*-sized font from the app's theme. The layout editor's **Palette** provides preconfigured `TextView`s named **Plain Text**, **Large Text**, **Medium Text** and **Small Text** (in the **Widgets** section) for various text sizes. The **Plain Text** `TextView` uses the theme's default font size. For the others, the IDE configures the `TextView`'s `textAppearance` property using the Material theme's styles for the corresponding font sizes.

Perform the following steps to add to the `GridLayout` an `EditText` and a `TextView` for receiving and displaying the bill amount:

1. This app allows you to enter only *nonnegative integers*, which the app divides by 100.0 to display the bill amount. The **Palette**'s **Text Fields** section provides *preconfigured* `EditText`s for various forms of input, including person names, passwords, e-mail addresses, phone numbers, times, dates and numbers. When the user interacts with an `EditText`, an appropriate keyboard is displayed, based on the `EditText`'s *input type*. From the **Palette**'s **Text Fields** section, drag and drop a **Number** `EditText` onto the `GridLayout` node in the **Component Tree** window—this creates an `EditText` with the `id` `editText` in the `GridLayout`. Change the `id` to `amountEditText`. The `EditText` is placed in the *first* column of the `GridLayout`'s *first* row. Set the `EditText`'s `layout:column` to 0 and the `layout:columnSpan` to 2—these settings ensure that the `TextView` spans both columns of row 0.
2. Drag a **Medium Text** `TextView` from the **Palette**'s **Widgets** section over the `amountEditText` in the **Component Tree** window—a horizontal black line appears below `amountEditText`, indicating that the `TextView` will be placed after `amountEditText`. The IDE creates a new `TextView` named `textView` and nests it in the `GridLayout` node. The default text "Medium Text" appears in the layout editor. You'll change this in *Step 5* (Section 3.4.5). Change the `TextView`'s `id` to `amountTextView`, then set the `layout:row` to 0, the `layout:column` to 0 and the `layout:columnSpan` to 2—these settings ensure that the `TextView` spans both columns of row 0, as you'll see once we change the `TextView`'s background color.

Step 2: Adding Views to the Second Row

Next, add the `percentTextView` and `percentSeekBar` to the `GridLayout` for displaying and selecting the tip percentage (be sure to set each view's `id` to the name we specify):

1. Drag a **Medium Text** `TextView` (`percentTextView`) from the **Palette**'s **Widgets** section over the `amountTextView` in the `GridLayout` node in the **Component Tree** window. The new view becomes the first view in row 1 (the second row).
2. Drag a **SeekBar** (`percentSeekBar`) from the **Palette**'s **Widgets** section over the `percentTextView` in the `GridLayout` node in the **Component Tree** window. The new view becomes the second view in row 1.

Step 3: Adding Views to the Third Row

Next, add the `tipLabelTextView` and the `tipTextView` to the `GridLayout` for displaying the tip amount:

1. Drag a **Medium TextView** (`tipLabelTextView`) over the `percentSeekBar` in the `GridLayout` node. The new view becomes the first view in row 2 (the third row).
2. Drag a **Medium TextView** (`tipTextView`) over the `tipLabelTextView` in the `GridLayout` node. The new view becomes the second view in row 2.

Step 4: Adding Views to the Fourth Row

Next, add the `totalLabelTextView` and the `totalTextView` to the `GridLayout` for displaying the tip amount:

1. Drag a **Medium TextView** (`totalLabelTextView`) over the `tipTextView` in the `GridLayout` node. This becomes the first view in row 3 (the fourth row).
2. Drag a **Medium TextView** (`totalTextView`) over the `totalLabelTextView` in the `GridLayout` node. This becomes the second view in row 3.

Reviewing the Layout So Far

The GUI and **Component Tree** window should now appear as shown in Fig. 3.4. The warning symbols shown in the layout editor and the **Component Tree** window will go away as you complete the GUI design in Section 3.4.5.

a) GUI design so far

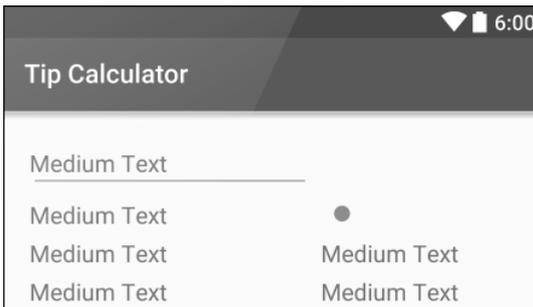
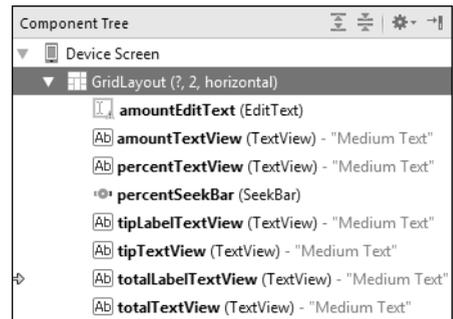
b) **Component Tree** window showing the **Tip Calculator**'s layout and views

Fig. 3.4 | GUI and the **Component Tree** window after adding the views to the `GridLayout`.

A Note Regarding the EditText's Virtual Keyboard

When the virtual keyboard is displayed, the device's back button (⏪) changes to a down button (⏴) that enables you to dismiss the keyboard. If you do so, the down button (⏴) changes to a back button (⏪) that you can touch to return to the previous `Activity`—possibly a prior app or the device's home screen.

Normally, you'd touch the `EditText` to redisplay the virtual keyboard. In this app, however, the `EditText` is hidden behind a `TextView`. If you were to dismiss this app's keyboard, you'd have to leave the app and return to it to redisplay the keyboard. We could programmatically force the keyboard to stay on the screen, but this would prevent the back

button from ever being displayed in this app. This, in turn, would prevent you from returning to the previous `Activity`—a basic Android feature that every user expects.

We used an Android virtual keyboard to demonstrate how to choose the keyboard displayed for a given `EditText`. Another approach would be to provide `Buttons` representing the digits 0–9 that always remain on the screen. We could handle their click events and use `String` manipulation rather than an `EditText` to keep track of the user input.

3.4.5 Customizing the Views

You'll now customize additional view properties. As you did in Section 2.5, you'll also create several `String`, dimension and color resources.

Step 5: Specifying Literal Text

Next, you'll specify the literal text for the `amountTextView`, `percentTextView`, `tipLabelTextView` and `totalLabelTextView`. When a `TextView`'s `text` property is empty, its `hint` property's value (if you specify one) is displayed—this property is commonly used with an `EditText` (a subclass of `TextView`) to help the user understand the `EditText`'s purpose. We're using it similarly in the `amountTextView` to tell the user to enter a bill amount:

1. In the **Component Tree**, select `amountTextView` and locate its `hint` property in the **Properties** window.
2. Click the ellipsis (...) button to the right of the property's value to display the **Resources** dialog.
3. In the dialog, click **New Resource**, then select **New String Value...** to display the **New String Value Resource** dialog and set the **Resource name** to `enter_amount` and **Resource value** to "Enter Amount". Leave the other settings and click **OK** to create the new `String` resource and set it as `amountTextView`'s `hint`.

Repeat these steps to set the `text` property for the `percentTextView`, `tipLabelTextView` and `totalLabelTextView` using the values shown in Fig. 3.5.

View	Resource name	Resource Value
<code>percentTextView</code>	<code>tip_percentage</code>	15%
<code>tipLabelTextView</code>	<code>tip</code>	Tip
<code>totalLabelTextView</code>	<code>total</code>	Total

Fig. 3.5 | String resource values and names.

Step 6: Right Aligning the TextViews in the Left Column

In Fig. 3.2, the `percentTextView`, `tipLabelTextView` and `totalLabelTextView` are right aligned. You can accomplish this for all three `TextViews` at once as follows:

1. Select the `percentTextView`.
2. Hold `Ctrl` on Windows/Linux or `Command` on Mac and click the `tipLabelTextView` and `totalLabelTextView`. Now all three `TextViews` are selected.
3. Expand the `layout:gravity` property's node and check the `right` checkbox.

Step 7: Configuring the `amountEditText`

In the final app, the `amountEditText` is *hidden* behind the `amountTextView` and is configured to allow only *digits* to be entered by the user. Select the `amountEditText` and set the following properties:

1. Set the **digits** property to `0123456789`—this allows *only* digits to be entered, even though the numeric keypad contains other characters, such as minus (-), comma (,) and period (.).
2. Set the **maxLength** property to 6. This restricts the bill amount to a maximum of *six* digits—so the largest supported bill amount is `9999.99`.

Step 8: Configuring the `amountTextView`

To complete the `amountTextView`'s formatting, select it and set the following properties:

1. Delete the default value of the **text** property ("`Medium Text`")—we'll programmatically display text here, based on the user's input.
2. Expand the **layout:gravity** property's node and set the **fill** to `horizontal`. This indicates that the `TextView` should occupy all remaining horizontal space in this `GridLayout` row.
3. Set the **background** property (which specifies the view's background color) to a new color resource named `amount_background` with the value `#BBDEFB`—a light blue color chosen from Google's material design color palette.
4. Add padding around the `TextView`. A view's **padding** specifies extra space around a view's content. The **all** property specifies that the padding amount should be applied to the top, right, bottom and left of the view's contents. You may also set the padding for each of these individually. Expand the **padding** property's node, click the **all** property, then click the ellipsis button. Create a new dimension resource named `textview_padding` with the value `12dp`. You'll use this resource again shortly.
5. Finally, add a shadow to the view by setting the **elevation** property to a new dimension resource named `elevation` with the value `4dp`. We chose this value for demonstration purposes to emphasize the shadow effect.

Step 9: Configuring the `percentTextView`

Notice that the `percentTextView` is aligned higher than the `percentSeekBar`. This looks better if it's *vertically centered*. To do this, expand the **layout:gravity** property's node, then set the center value to `vertical`. Recall that you previously set the **layout:gravity** to `right`. The combination of these settings appears in the layout XML as

```
android:layout_gravity="center_vertical|right"
```

A *vertical bar* (|) is used to separate multiple **layout:gravity** values—in this case indicating that the `TextView` should be *centered vertically* and *right aligned* within the grid cell.

Step 10: Configuring the `percentSeekBar`

Select `percentSeekBar` and configure the following properties:

1. By default, a `SeekBar`'s range is 0 to 100 and its current value is indicated by its **progress** property. This app allows tip percentages from 0 to 30 and specifies a

default of 15 percent. Set the `SeekBar`'s `max` property to 30 and the `progress` property to 15.

2. Expand the `layout:gravity` property's node and set the `fill` to `horizontal` so the `SeekBar` occupies all horizontal space in the `SeekBar`'s `GridLayout` column.
3. Set the `layout:height` property to a new dimension resource (`seekbar_height`) with the value `40dp` to increase vertical space in which the `SeekBar` is displayed.

Step 11: Configuring the `tipTextView` and `totalTextView`

To complete the formatting of the `tipTextView` and `totalTextView`, select both and set the following properties:

1. Delete the default value of the `text` property ("Medium Text")—we'll programmatically display the calculated tip and total.
2. Expand the `layout:gravity` property's node and set the `fill` to `horizontal` so each `TextView` occupies all horizontal space in the `TextViews`' `GridLayout` column.
3. Set the `background` property to a new color resource named `result_background` with the value `#FFE0B2`—a light orange color chosen from Google's material design color palette.
4. Set the `gravity` property to `center` so the calculated tip and total amounts will be centered within these `TextViews`.
5. Expand the `padding` property's node, click the ellipsis button for the `all` value, then select the dimension resource named `textView_padding` that you created previously for the `amountTextView`.
6. Finally, add a shadow to each view by setting the `elevation` property to the `elevation` dimension resource you created earlier.

3.5 Default Theme and Customizing Theme Colors

Each app has a theme that defines the default look-and-feel of the standard views you use. The theme is specified in the app's `AndroidManifest.xml` file (Section 3.7). You can customize aspects of the theme, such those that define an app's color scheme, by defining **style** resources in the `styles.xml` file located in the in the app's `res/values` folder.

3.5.1 parent Themes

The `style.xml` resource file contains a `style` with the name "AppTheme" that's referenced from the app's `AndroidManifest.xml` file to specify the app's theme. This `style` also specifies a parent theme, which is similar to a superclass in Java—the new `style` inherits its parent theme's attributes and their default values. Just as in a Java subclass, a `style` can override parent theme attributes with values customized for specific apps. A company might do this, for example, to use the company's branding colors. We'll use this concept in Section 3.5.2 to customize three colors used in the app's theme.

As we mentioned previously, Android Studio's app templates now include support for the `AppCompat` libraries that enable you to use newer Android features in older Android versions. By default, Android Studio sets the parent theme to

```
Theme.AppCompat.Light.DarkActionBar
```

one of several predefined themes from the AppCompat library—apps that use this theme have a light background, except for the dark app bar at the top of the app. Each AppCompat theme uses Google’s material design recommendations to style your apps’ GUIs.

3.5.2 Customizing Theme Colors

Section 3.3.11 discussed where a theme’s primary, dark primary and accent colors are applied in an app’s on-screen elements. In this section, you’ll use the new Android Studio **Theme Editor** to change the app’s primary, dark primary and accent colors, thus overriding the values of the `android:colorPrimary`, `android:colorPrimaryDark` and `android:colorAccent` theme attributes shown in Fig. 3.6. These are three of many theme attributes you can override. For the complete list, visit:

<http://developer.android.com/reference/android/R.attr.html>

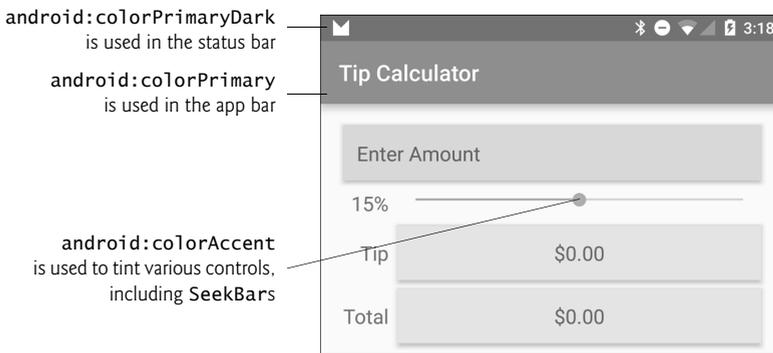


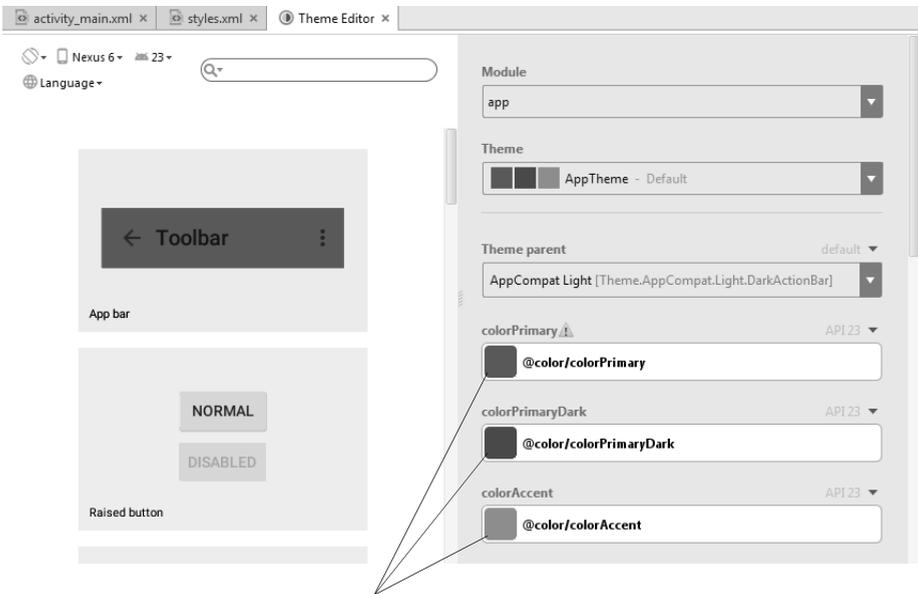
Fig. 3.6 | Theme attributes for the primary, primary dark and accent colors.

Modifying the Theme’s Primary, Dark Primary and Accent Colors

To customize the colors:

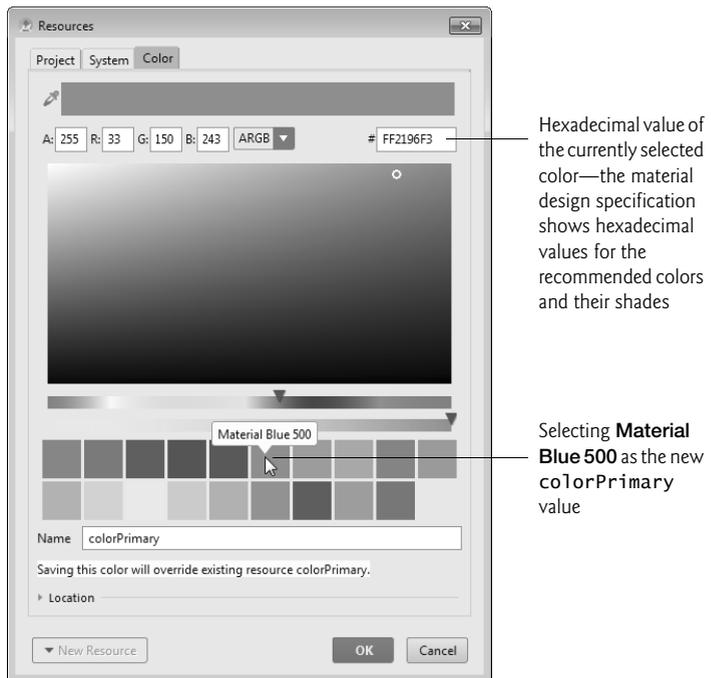
1. Open `styles.xml`. In the editor’s upper-right corner, click the **Open editor** link to display the **Theme Editor** (Fig. 3.7) showing the current colors for `colorPrimary` (dark blue), `colorPrimaryDark` (a darker shade of `colorPrimary`) and `colorAccent` (bright pink)—these are the default colors specified in Android Studio’s **Empty Activity** app template. For this app, we’ll change `colorPrimary` and `colorPrimaryDark` to lighter blues and change `colorAccent` to orange.
2. Customize the app’s `colorPrimary` value by clicking its color swatch (Fig. 3.7) to display the **Resources** dialog (Fig. 3.8). In the dialog, click the **Material Blue 500** color swatch, then click **OK** to change `colorPrimary`’s value—hovering the mouse cursor over a color swatch displays its color name in a tooltip. The number **500** represents a particular shade of the **Material Blue** color. Shades of each color range from 50 (a light shade) to 900 (a dark shade)—you can view samples of each color’s shades at

<https://www.google.com/design/spec/style/color.html#color-color-palette>



Color swatches for the theme's colorPrimary, colorPrimaryDark and colorAccent attributes

Fig. 3.7 | Theme Editor shows styled view previews on the left and theme attributes on the right.



Hexadecimal value of the currently selected color—the material design specification shows hexadecimal values for the recommended colors and their shades

Selecting **Material Blue 500** as the new colorPrimary value

Fig. 3.8 | Selecting the **Material Blue 500** color swatch for colorPrimary.

- Next, click the `colorPrimaryDark` color swatch in the **Theme Editor** to display the **Resources** dialog. The **Theme Editor** recognizes the new `colorPrimary` value and automatically displays a color swatch containing the recommended darker `colorPrimary` shade you should use for `colorPrimaryDark`—in this case, **Material Blue 700**. Click that color swatch (Fig. 3.9), then click **OK**.

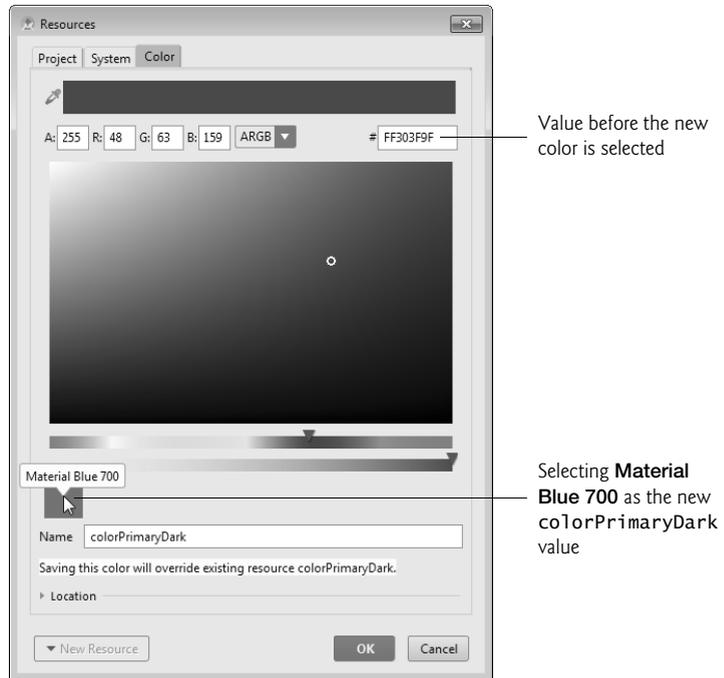


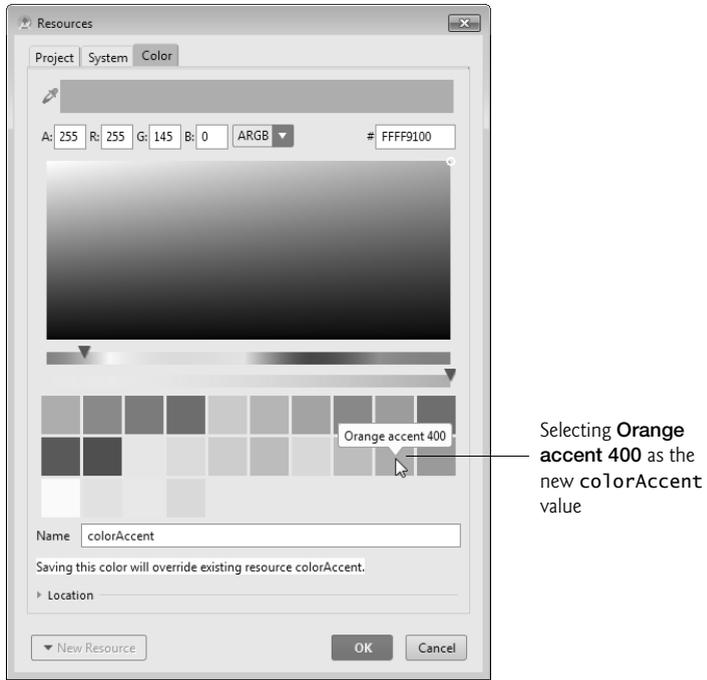
Fig. 3.9 | Selecting the **Material Blue 700** color swatch for `colorPrimaryDark`.

- Next, click the `colorAccent` color swatch in the **Theme Editor** to display the **Resources** dialog. Again, the **Theme Editor** recognizes that you changed the `colorPrimary` value and displays swatches for various complementary accent colors. In the dialog, click the **Orange accent 400** color swatch, then click **OK** to change `colorAccent`'s value (Fig. 3.10), then click **OK**.

You've now completed the app's design, which should appear as shown in Fig. 3.11.

3.5.3 Common View Property Values as Styles

As you'll see in later apps, `style` resources can define common property values that should be applied to multiple views. You apply a `style` resource to a given view by setting its `style` property. Any subsequent changes you make to a `style` are automatically applied to all views using the `style`. For example, consider the `tipTextView` and `totalTextView` that we configured identically in Step 11 of Section 3.4.5. We could have defined a `style` resource specifying the `layout:gravity`, `background`, `gravity`, `padding` and `elevation` properties' values, then set both `TextView`s' `style` properties to the same `style` resource.



Selecting **Orange accent 400** as the new `colorAccent` value

Fig. 3.10 | Selecting the **Orange accent 400** color swatch for `colorAccent`.

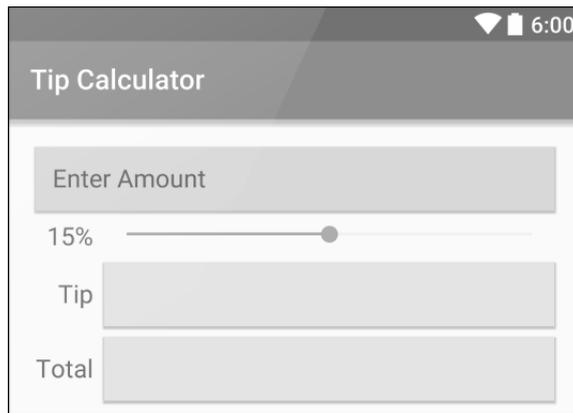


Fig. 3.11 | Completed design.

3.6 Adding the App's Logic

Class `MainActivity` (Figs. 3.12–3.18) implements the **Tip Calculator** app's logic. It calculates the tip and total bill amounts, then displays them in locale-specific currency format. To view the file, in the Project window, expand the `app/Java/com.deitel.tipcalcula-`

tor node and double click `MainActivity.java`. You'll need to enter most of the code in Figs. 3.12–3.18.

3.6.1 package and import Statements

Figure 3.12 shows the package statement and import statements in `MainActivity.java`. The package statement in line 3 was inserted when you created the project. When you open a Java file in the IDE, the import statements are collapsed—one is displayed with a  to its left. You can click the  to see the complete list of import statements.

```

1 // MainActivity.java
2 // Calculates a bill total based on a tip percentage
3 package com.deitel.tipcalculator;
4
5 import android.os.Bundle; // for saving state information
6 import android.support.v7.app.AppCompatActivity; // base class
7 import android.text.Editable; // for EditText event handling
8 import android.text.TextWatcher; // EditText listener
9 import android.widget.EditText; // for bill amount input
10 import android.widget.SeekBar; // for changing the tip percentage
11 import android.widget.SeekBar.OnSeekBarChangeListener; // SeekBar listener
12 import android.widget.TextView; // for displaying text
13
14 import java.text.NumberFormat; // for currency formatting
15

```

Fig. 3.12 | `MainActivity`'s package and import statements.

Lines 5–14 import the classes and interfaces the app uses:

- Class `Bundle` of package `android.os` (line 5) stores key–value pairs of information—typically representing an app's state or data that needs to be passed between activities. When another app is about to appear on the screen—e.g., when the user *receives a phone call* or *launches another app*—Android gives the currently executing app the opportunity to *save its state* in a `Bundle`. The Android runtime might subsequently kill the app—e.g., to reclaim its memory. When the app returns to the screen, the Android runtime passes the `Bundle` of the previously saved state to `Activity` method `onCreate` (Section 3.6.4). Then, the app can use the saved state to return the app to the state it was in when another app became active. We'll use `Bundles` in Chapter 8 to pass data between activities.
- Class `AppCompatActivity` of package `android.support.v7.app` (line 6) provides the basic *lifecycle methods* of an app—we'll discuss these shortly. `AppCompatActivity` is an indirect subclass of `Activity` (package `android.app`) that supports using newer Android features apps running on current and older Android platforms.
- Interface `Editable` of package `android.text` (line 7) allows you to modify the content and markup of text in a GUI.
- You implement interface `TextWatcher` of package `android.text` (line 8) to respond to events when the user changes the text in an `EditText`.

- Package `android.widget` (lines 9, 10 and 12) contains the *widgets* (i.e., views) and layouts that are used in Android GUIs. This app uses `EditText` (line 9), `SeekBar` (line 10) and `TextView` (line 12) widgets.
- You implement interface `SeekBar.OnSeekBarChangeListener` of package `android.widget` (line 11) to respond to the user moving the `SeekBar`'s *thumb*.
- Class `NumberFormat` of package `java.text` (line 14) provides numeric formatting capabilities, such as *locale-specific* currency and percentage formats.

3.6.2 MainActivity Subclass of AppCompatActivity

Class `MainActivity` (Figs. 3.13–3.18) is the **Tip Calculator** app's `Activity` subclass. When you created the `TipCalculator` project, the IDE generated this class as a subclass of `AppCompatActivity` (an indirect subclass of `Activity`) and provided an override of class `Activity`'s inherited `onCreate` method (Fig. 3.15). Every `Activity` subclass *must* override this method. We'll discuss `onCreate` shortly.

```

16 // MainActivity class for the Tip Calculator app
17 public class MainActivity extends Activity {
18

```

Fig. 3.13 | Class `MainActivity` is a subclass of `Activity`.

3.6.3 Class Variables and Instance Variables

Figure 3.14 declares class `MainActivity`'s variables. The `NumberFormat` objects (lines 20–23) are used to format currency values and percentages, respectively. `NumberFormat`'s static method `getCurrencyInstance` returns a `NumberFormat` object that formats values as currency using the device's locale. Similarly, static method `getPercentInstance` formats values as percentages using the device's locale.

```

19 // currency and percent formatter objects
20 private static final NumberFormat currencyFormat =
21     NumberFormat.getCurrencyInstance();
22 private static final NumberFormat percentFormat =
23     NumberFormat.getPercentInstance();
24
25 private double billAmount = 0.0; // bill amount entered by the user
26 private double percent = 0.15; // initial tip percentage
27 private TextView amountTextView; // shows formatted bill amount
28 private TextView percentTextView; // shows tip percentage
29 private TextView tipTextView; // shows calculated tip amount
30 private TextView totalTextView; // shows calculated total bill amount
31

```

Fig. 3.14 | `MainActivity` class's instance variables.

The bill amount entered by the user into `amountEditText` will be read and stored as a `double` in `billAmount` (line 25). The tip percentage (an integer in the range 0–30) that the user sets by moving the `SeekBar` *thumb* will be divided by 100.0 to create a `double` for

use in calculations, then stored in percent (line 26). For example, if you select 25 with the SeekBar, percent will store 0.25, so the app will multiply the bill amount by 0.25 to calculate the 25% tip.



Software Engineering Observation 3.3

For precise monetary calculations, use class `BigDecimal` (package `java.math`)—rather than type `double`—to represent the monetary amounts and perform calculations.

Line 27 declares the `TextView` that displays the currency-formatted bill amount. Line 28 declares the `TextView` that displays the tip percentage, based on the SeekBar *thumb's* position (see the 15% in Fig. 3.1(a)). The variables in line 29–30 will refer to the `TextView`s in which the app displays the calculated tip and total.

3.6.4 Overriding Activity Method `onCreate`

The `onCreate` method (Fig. 3.15)—which is *autogenerated* with lines 33–36 when you create the app's project—is called by the system when an `Activity` is *started*. Method `onCreate` typically initializes the `Activity`'s instance variables and views. This method should be as simple as possible so that the app *loads quickly*. In fact, if the app takes longer than five seconds to load, the operating system will display an ANR (**A**pplication **N**ot **R**esponding) dialog—giving the user the option to *forcibly terminate the app*. You'll learn how to prevent this problem in Chapter 9.

```

32 // called when the activity is first created
33 @Override
34 protected void onCreate(Bundle savedInstanceState) {
35     super.onCreate(savedInstanceState); // call superclass's version
36     setContentView(R.layout.activity_main); // inflate the GUI
37
38     // get references to programmatically manipulated TextViews
39     amountTextView = (TextView) findViewById(R.id.amountTextView);
40     percentTextView = (TextView) findViewById(R.id.percentTextView);
41     tipTextView = (TextView) findViewById(R.id.tipTextView);
42     totalTextView = (TextView) findViewById(R.id.totalTextView);
43     tipTextView.setText(currencyFormat.format(0)); // set text to 0
44     totalTextView.setText(currencyFormat.format(0)); // set text to 0
45
46     // set amountEditText's TextWatcher
47     EditText amountEditText =
48         (EditText) findViewById(R.id.amountEditText);
49     amountEditText.addTextChangedListener(amountEditTextWatcher);
50
51     // set percentSeekBar's OnSeekBarChangeListener
52     SeekBar percentSeekBar =
53         (SeekBar) findViewById(R.id.percentSeekBar);
54     percentSeekBar.setOnSeekBarChangeListener(seekBarListener);
55 }
56

```

Fig. 3.15 | Overriding Activity method `onCreate`.

onCreate's Bundle Parameter

During the app's execution, the user could change the device's configuration—for example, by *rotating the device, connecting to a Bluetooth keyboard or sliding out a hard keyboard*. For a good user experience, the app should continue operating smoothly through such configuration changes. When the system calls `onCreate`, it passes a `Bundle` argument containing the `Activity`'s saved state, if any. Typically, you save state in `Activity` methods `onPause` or `onSaveInstanceState` (demonstrated in later apps). Line 35 calls the superclass's `onCreate` method, which is *required* when overriding `onCreate`.

Generated R Class Contains Resource IDs

As you build your app's GUI and add *resources* (such as strings in the `strings.xml` file or views in the `activity_main.xml` file) to your app, the IDE generates a class named `R` that contains *nested classes* representing each type of resource in your project's `res` folder. The nested classes are declared `static`, so that you can access them in your code with `R.ClassName`. Within class `R`'s nested classes, the IDE creates `static final int` constants that enable you to refer to your app's resources programmatically (as we'll discuss momentarily). Some of the nested classes in class `R` include

- class `R.drawable`—contains constants for any `drawable` items, such as *images*, that you put in the various `drawable` folders in your app's `res` folder
- class `R.id`—contains constants for the *views* in your *XML layout files*
- class `R.layout`—contains constants that represent each *layout file* in your project (such as, `activity_main.xml`), and
- class `R.string`—contains constants for each `String` in the `strings.xml` file.

Inflating the GUI

The call to `setContentView` (line 36) receives the constant `R.layout.activity_main` which indicates the XML file that represents `MainActivity`'s GUI—in this case, the constant represents the `activity_main.xml` file. Method `setContentView` uses this constant to load the corresponding XML document, which Android parses and converts into the app's GUI. This process is known as *inflating the GUI*.

Getting References to the Widgets

Once the layout is *inflated*, you can *get references to the individual widgets* so that you can interact with them programmatically. To do so, you use class `Activity`'s `findViewById` method. This method takes an `int` constant representing a specific view's `id` and returns a reference to the view. The name of each view's `R.id` constant is determined by the component's `id` property that you specified when designing the GUI. For example, `amountEditText`'s constant is `R.id.amountEditText`.

Lines 39–42 obtain references to the `TextViews` that we change programmatically in the app. Line 39 obtains a reference to the `amountTextView` that's updated when the user enters the bill amount. Line 40 obtains a reference to the `percentTextView` that's updated when the user changes the tip percentage. Lines 41–42 obtain references to the `TextViews` where the calculated tip and total are displayed.

Displaying Initial Values in the TextViews

Lines 43–44 set `tipTextView`'s and `totalTextView`'s text to 0 in a *locale-specific* currency format by calling the `CurrencyFormat` object's `format` method. The text in each of these `TextView`s will change as the user enters the bill amount.

Registering the Event Listeners

Lines 47–49 get the `amountEditText` and call its `addTextChangedListener` method to register the `TextWatcher` object that responds to events generated when the user changes the `EditText`'s contents. We define this listener (Fig. 3.18) as an anonymous-inner-class object and assign it to the `amountEditTextWatcher` instance variable. Though we could have defined the anonymous inner class in place of `amountEditTextWatcher` in line 49, we chose to define it later in the class so that the code is easier to read.



Software Engineering Observation 3.4

Rather than defining anonymous inner classes in large methods, define them as private final instance variables to make your code easier to debug, modify and maintain.

Lines 52–53 get a reference to the `percentSeekBar`. Line 54 calls the `SeekBar`'s `setOnSeekBarChangeListener` method to register the `OnSeekBarChangeListener` object that responds to *events* generated when the user moves the `SeekBar`'s thumb. Figure 3.17 defines this listener as an anonymous-inner-class object that's assigned to the instance variable `seekBarListener`.

Note Regarding Android 6 Data Binding

Android now has a **Data Binding support library** that you can use with Android apps targeting Android 2.1 (API level 7) and higher. You now can include in your layout XML files data-binding expressions that manipulate Java objects and dynamically update data in your apps' user interfaces.

In addition, each layout XML file that contains views with `ids` has a corresponding autogenerated class. For each view with an `id`, the class has a `public final` instance variable referencing that view. You can create an instance of this “Binding” class to replace all calls to `findViewById`, which can greatly simplify your `onCreate` methods in `Activity` and `Fragment` classes with complex user interfaces. Each instance variable's name is the `id` specified in the layout for the corresponding view. The “Binding” class's name is based on the layout's name—for `activity_main.xml`, the class name is `ActivityMainBinding`.

At the time of this writing, the Data Binding library is an early beta release that's subject to substantial changes, both in the syntax of data-binding expressions and in the Android Studio tool support. You can learn more about Android data binding at

<https://developer.android.com/tools/data-binding/guide.html>

3.6.5 MainActivity Method calculate

Method `calculate` (Fig. 3.16) is called by the `EditText`'s and `SeekBar`'s listeners to update the tip and total `TextView`s each time the user *changes* the bill amount. Line 60 displays the tip percentage in the `percentTextView`. Lines 63–64 calculate the tip and total, based on the `billAmount`. Lines 67–68 display the amounts in currency format.

```

57 // calculate and display tip and total amounts
58 private void calculate() {
59     // format percent and display in percentTextView
60     percentTextView.setText(percentFormat.format(percent));
61
62     // calculate the tip and total
63     double tip = billAmount * percent;
64     double total = billAmount + tip;
65
66     // display tip and total formatted as currency
67     tipTextView.setText(currencyFormat.format(tip));
68     totalTextView.setText(currencyFormat.format(total));
69 }
70

```

Fig. 3.16 | MainActivity Method calculate.

3.6.6 Anonymous Inner Class That Implements Interface OnSeekBarChangeListener

Lines 72–87 (Fig. 3.17) create the *anonymous-inner-class* object that responds to percent-SeekBar’s *events*. The object is assigned to the instance variable `seekBarListener`. Line 54 (Fig. 3.15) registered `seekBarListener` as `percentSeekBar`’s `OnSeekBarChangeListener` *event-handling* object. For clarity, we define all but the simplest event-handling objects in this manner so that we do not clutter the `onCreate` method with this code.

```

71 // listener object for the SeekBar's progress changed events
72 private final OnSeekBarChangeListener seekBarListener =
73     new OnSeekBarChangeListener() {
74         // update percent, then call calculate
75         @Override
76         public void onProgressChanged(SeekBar seekBar, int progress,
77             boolean fromUser) {
78             percent = progress / 100.0; // set percent based on progress
79             calculate(); // calculate and display tip and total
80         }
81
82         @Override
83         public void onStartTrackingTouch(SeekBar seekBar) { }
84
85         @Override
86         public void onStopTrackingTouch(SeekBar seekBar) { }
87     };
88

```

Fig. 3.17 | Anonymous inner class that implements interface `OnSeekBarChangeListener`.

Overriding Method onProgressChanged of Interface OnSeekBarChangeListener

Lines 75–86 (Fig. 3.17) implement interface `OnSeekBarChangeListener`’s methods. Method `onProgressChanged` is called whenever the `SeekBar`’s thumb position changes. Line 78 calculates the percent value using the method’s `progress` parameter—an `int` rep-

representing the `SeekBar`'s *thumb* position. We divide this by 100.0 to get the percentage. Line 79 calls method `calculate` to recalculate and display the tip and total.

Overriding Methods `onStartTrackingTouch` and `onStopTrackingTouch` of Interface `OnSeekBarChangeListener`

Java requires that you override *every* method in an interface that you implement. This app does not need to know when the user starts moving the `SeekBar`'s thumb (`onStartTrackingTouch`) or stops moving it (`onStopTrackingTouch`), so we simply provide an empty body for each (lines 82–86) to fulfill the interface contract.

Android Studio Tools for Overriding Methods

Android Studio can create for you empty methods that override inherited methods from the class's superclasses or that implement interface methods. When you place the cursor in a class's body, then select the **Code > Override Methods...** menu option, the IDE displays a **Select Methods to Override/Implement** dialog that lists every method you can override in the current class. This list includes all the inherited methods in the class's hierarchy and the methods of any interfaces implemented throughout the class's hierarchy.



Error-Prevention Tip 3.2

Using Android Studio's **Code > Override Methods...** menu option helps you write code faster and with fewer errors.

3.6.7 Anonymous Inner Class That Implements Interface `TextWatcher`

Lines 90–114 of Fig. 3.18 create an *anonymous-inner-class* object that responds to `amountEditText`'s *events* and assign it to the instance variable `amountEditTextWatcher`. Line 49 (Fig. 3.15) registered this object to *listen* for `amountEditText`'s events that occur when the text changes.

```

89 // listener object for the EditText's text-changed events
90 private final TextWatcher amountEditTextWatcher = new TextWatcher() {
91     // called when the user modifies the bill amount
92     @Override
93     public void onTextChanged(CharSequence s, int start,
94         int before, int count) {
95
96         try { // get bill amount and display currency formatted value
97             billAmount = Double.parseDouble(s.toString()) / 100.0;
98             amountTextView.setText(currencyFormat.format(billAmount));
99         }
100        catch (NumberFormatException e) { // if s is empty or non-numeric
101            amountTextView.setText("");
102            billAmount = 0.0;
103        }
104
105        calculate(); // update the tip and total TextViews
106    }
107

```

Fig. 3.18 | Anonymous inner class that implements interface `TextWatcher`. (Part 1 of 2.)

```

108     @Override
109     public void afterTextChanged(Editable s) { }
110
111     @Override
112     public void beforeTextChanged(
113         CharSequence s, int start, int count, int after) { }
114 };
115 }

```

Fig. 3.18 | Anonymous inner class that implements interface `TextWatcher`. (Part 2 of 2.)

Overriding Method `onTextChanged` of Interface `TextWatcher`

The `onTextChanged` method (lines 92–106) is called whenever the text in the `amountEditText` is *modified*. The method receives four parameters. In this example, we use only `CharSequence s`, which contains a copy of `amountEditText`'s text. The other parameters indicate that the count characters starting at `start` *replaced* previous text of length `before`.

Line 97 converts the user input from `amountEditText` to a `double`. We allow users to enter only whole numbers in pennies, so we divide the converted value by 100.0 to get the actual bill amount—e.g., if the user enters 2495, the bill amount is 24.95. Line 98 displays the updated bill amount. If an exception occurs, lines 101–102 clear the `amountTextView` and set the `billAmount` to 0.0. Lines 105 calls `calculate` to recalculate and display the tip and total, based on the current bill amount.

Other Methods of the `amountEditTextWatcher TextWatcher`

This app does *not* need to know what changes are about to be made to the text (`beforeTextChanged`) or that the text has already been changed (`afterTextChanged`), so we simply override each of these `TextWatcher` interface methods with an *empty* body (lines 108–113) to fulfill the interface contract.

3.7 AndroidManifest.xml

In this section, you'll modify the `AndroidManifest.xml` file to specify that this app's `Activity` supports only a device's portrait orientation and that the virtual keyboard should always be displayed when the `Activity` first appears on the screen or navigates back to the `Activity`. To open the manifest, double click `AndroidManifest.xml` in the **Project** window's `manifests` folder. Figure 3.19 shows the completed manifest with our changes highlighted—the rest of the file was autogenerated by Android Studio when we created the app's project. We'll discuss some aspects of the manifest here. For a list of all the elements a manifest may contain, their attributes and their values, visit

<http://developer.android.com/guide/topics/manifest/manifest-intro.html>

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.deitel.tipcalculator" >
4

```

Fig. 3.19 | `AndroidManifest.xml` contents. (Part 1 of 2.)

```

5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="@string/app_name"
9         android:supportsRtl="true"
10        android:theme="@style/AppTheme" >
11        <activity
12            android:name=".MainActivity"
13            android:label="@string/app_name"
14            android:screenOrientation="portrait"
15            android:windowSoftInputMode="stateAlwaysVisible">
16            <intent-filter>
17                <action android:name="android.intent.action.MAIN" />
18
19                <category android:name="android.intent.category.LAUNCHER" />
20            </intent-filter>
21        </activity>
22    </application>
23
24 </manifest>

```

Fig. 3.19 | AndroidManifest.xml contents. (Part 2 of 2.)

3.7.1 manifest Element

The **manifest** element (lines 2–24) indicates that this XML file’s contents represent the app’s manifest. This element’s package attribute specifies the app’s Java package name that was configured when you created the app’s project (Section 3.4.2). Recall that for apps you submit to the Google Play store, the package name is used as the app’s unique identifier.

3.7.2 application Element

The **manifest** element’s nested **application** element (lines 5–21) specifies attributes of the application, including

- **android:allowBackup**—Whether or not the app’s data should be backed up automatically by Android so that the data can be restored to the device or a new device at a later time.
- **android:icon**—The app icon that you touch in the launcher to execute the app.
- **android:label**—The app’s name that’s typically displayed below the icon in the launcher and often displayed in the app bar when the app is executing.
- **android:supportsRtl**—Whether or not the app’s interface can be flipped horizontally to support right-to-left languages like Arabic and Hebrew.
- **android:theme**—The theme that determines the default look-and-feel of the app’s views.

Elements nested in the **application** element define the app’s components, such as its activities.

3.7.3 activity Element

The application element's nested **activity** element (lines 10–20) describes an **Activity**. An app can have many activities, one of which is designated as the **Activity** that's displayed when the user touches the app's icon in the launcher to execute the app. Each activity element specifies at least the following attributes:

- **android:name**—The **Activity**'s class name. The notation ".MainActivity" is shorthand for "com.deitel.MainActivity" (where *com.deitel* is the reverse of the domain name you specified when creating the app's project).
- **android:label**—The **Activity**'s name. This is often displayed in the app bar when the **Activity** is on the screen. For single **Activity** apps, this name is typically the same as the app's name.

For **MainActivity**, we added the following attributes:

- **android:screenOrientation**—In general, most apps should support *both* portrait and landscape orientations. In *portrait* orientation, the device's longer dimension is vertical. In *landscape orientation*, the device's longer dimension is horizontal. In the **Tip Calculator** app, rotating the device to landscape orientation on a typical phone would cause the numeric keypad to obscure most of the **Tip Calculator**'s GUI. For this reason, we set this property to "portrait" to support *only* portrait orientation.
- **android:windowSoftInputMode**—In the **Tip Calculator** app, the soft keypad should be displayed immediately when the app executes and should reappear each time the user returns to the **Tip Calculator** app. For this reason we set this property to "stateAlwaysVisible". This will *not* display the soft keyboard if a hard keyboard is present.

3.7.4 intent-filter Element

Intents are Android's mechanism for communicating between executable components—such as activities, background services and the operating system. You *state your intent*, then Android uses **intent messaging** to coordinate the executable components to accomplish what you intend to do. This *loose coupling* makes it easy to mix and match parts of different applications. You tell Android what you want to accomplish, then let Android find the installed applications with activities that can handle the task.

Inter-App Communication

One example of how **Intents** are used is *coordinating efforts between separate apps*. Consider how photo sharing can be handled in Android:

- Most social-networking Android apps provide their own photo-sharing capabilities. Each app can advertise in its manifest its specific **Activity** that uploads a photo to the user's account.
- Other apps can use these photo-sharing capabilities, rather than implementing their own. For example, a photo-editing app can provide a **Share Photo** option. The app can respond to a user's photo-sharing request by *stating its intent* to share a photo—that is, creating a photo-sharing **Intent** and passing it to Android.

- Android looks at the `Intent` to determine which installed applications provide activities that can share photos.
- If there's only *one* such app, Android executes that app's photo-sharing `Activity`.
- If there are *many* such apps, Android displays a list of apps and asks the *user* to *decide* which app's photo-sharing `Activity` should execute.

A key benefit of this loosely coupled approach is that the photo-editing app's developer does not need to incorporate support for every possible social-networking site. By issuing a photo-sharing `Intent`, the app automatically supports *any* app that declares a photo-sharing `Activity` in its manifest, including those apps the user has already installed and any the user chooses to install in the future. For a list of the items that can be used with `Intents`, visit

<http://developer.android.com/reference/android/content/Intent.html#constants>

Executing Apps

Another example of how `Intents` are used is in *launching activities*. When you touch an app's icon in the device's launcher app, your intent is to execute the app. In this case, the launcher issues an `Intent` to execute that app's main `Activity` (discussed momentarily). Android responds to this `Intent` by launching the app and executing the specific `Activity` designated in the app's manifest as the main `Activity`.

Determining Which Activity to Execute

Android uses information in the manifest to determine the activities that can respond to `Intents` and which `Intents` each `Activity` can handle. In the manifest, the `activity` element's nested `intent-filter` element (Fig. 3.19, lines 16–20) determines which `Intent` types can launch an `Activity`. If an `Intent` matches only one `Activity`'s `intent-filter`, Android executes that `Activity`. If there are multiple matches, Android presents a list from which the user can choose an app, then executes the appropriate `Activity` in that app.

Android also passes the `Intent` to the `Activity`, because an `Intent` often contains data the `Activity` can use to perform its task. For example, a photo-editing app can include in a share-photo `Intent` the specific photo to share.

The `intent-filter` element must contain one or more `action` elements. The action `"android.intent.action.MAIN"` in line 17 of Fig. 3.19 indicates that `MainActivity` is the `Activity` to execute when the app launches. The optional `category` element in line 19 specifies what initiates the `Intent`—for `"android.intent.category.LAUNCHER"`, it's the device's launcher. This category also indicates that the `Activity` should appear as an icon in the device's launcher with the icons for the user's other installed apps.

We'll discuss and program with `Intents` in the next chapter. For more information on `Intents` and `Intent` filters, visit

<http://developer.android.com/guide/components/intents-filters.html>

3.8 Wrap-Up

In this chapter, you created the interactive **Tip Calculator** app. We discussed the app's capabilities, then you test-drove it to calculate the tip and total, based on the bill amount.

You built the app's GUI using Android Studio's layout editor, **Component Tree** window and **Properties** window. You also edited the layout's XML and used the **Theme Editor** to customize the `Theme.AppCompat.Light.DarkActionBar` theme's primary, dark primary and accent colors that were set by the IDE when you created the project. We presented the code for class `MainActivity`, a subclass of `AppCompatActivity` (and an indirect subclass of `Activity`) that defined the app's logic.

In the app's GUI, you used a `GridLayout` to arrange the views into rows and columns. You displayed text in `TextViews` and received input from an `EditText` and a `SeekBar`.

The `MainActivity` class required many Java object-oriented programming capabilities, including classes, objects, methods, interfaces, anonymous inner classes and inheritance. We explained the notion of inflating the GUI from its XML file into its screen representation. You learned about Android's `Activity` class and part of the `Activity` lifecycle. In particular, you overrode the `onCreate` method to initialize the app when it's launched. In the `onCreate` method, you used `Activity` method `findViewById` to get references to each of the views the app interacts with programmatically. You defined an anonymous inner class that implements the `TextWatcher` interface so the app can calculate new tips and totals as the user enters the bill amount in the `EditText`. You also defined an anonymous inner class that implements the `OnSeekBarChangeListener` interface so the app can calculate a new tip and total as the user changes the tip percentage by moving the `SeekBar`'s thumb.

Finally, you edited the `AndroidManifest.xml` file to specify that the `MainActivity` supports only portrait orientation and that the `MainActivity` should always display the keypad. We also discussed the other elements that Android Studio placed in the manifest when you created the project.

In Chapter 4, you'll build the **Flag Quiz** app in which the user is shown a graphic of a country's flag and must guess the country from 2, 4, 6 or 8 choices. You'll use a menu and checkboxes to customize the quiz, specifying the number of guess options and limiting the flags to specific regions of the world.

The top of the page features a decorative graphic consisting of a grid of various small icons, including symbols for Wi-Fi, a home button, a magnifying glass, and other common Android UI elements. The icons are arranged in a pattern that recedes into the distance, creating a 3D effect. The word "Index" is centered over this graphic in a large, bold, black serif font.

Index

Numerics

100 Destinations 5

A

accelerometer 17

 listening 190

accelerometer sensor 174, 192

access Android services 142

Accessibility

 TalkBack 36, 38, 66

 TalkBack localization 71

accessibility 32, 36, 38, 66

 contentDescription
 property 67

 Explore by Touch 9, 36, 38,
 66

Accessibility APIs 9

accessing Android content
 providers 15

action bar 77

action element of the manifest
 file 103

ACTION_SEND constant of class
 Intent 313

ACTION_VIEW constant of
 class Intent 311

Activity

 states 77

activity 76

Activity class 76, 77, 93, 94

 findFragmentById method
 114, 143

 getFragmentManager
 method 114, 159

 getMenuInflater method
 144

 getResources method 143,
 144

 getString method with
 multiple arguments 313

Activity class (cont.)

 getSystemService method
 190

 host a Fragment 114

 lifecycle methods 173, 220
 onCreate method 77, 95,
 173

 onCreateOptionsMenu
 method 113, 144

 onDestroy method 173,
 220

 onOptionsItemSelected
 method 113, 145

 onPause method 173

 onResume method 173

 onStart method 143, 173

 onStop method 173

 runOnUiThread method 247

 sent to background 226

 setContentView method 96

 setRequestedOrientation
 method 143

 setVolumeControlStream
 method 221, 226

<activity> element

activity element in

 AndroidManifest.xml 102

Activity Not Responding

 (ANR) dialog 263, 330

Activity templates 43

activity_main.xml 49

ActivityNotFoundException
 class 119, 296

Adapter class 263

AdapterView class 263

add method of class

 FragmentTransaction 356

addCallback method of class
 SurfaceHolder 240

addItemDecoration method of
 class RecyclerView 307

addPreferencesFromResource
 method of class

 PreferenceFragment 161

addToBackStack method of
 class FragmentTransaction
 357

adjustViewBounds property of
 an ImageView 135

AdMob 392

advertising revenue 392

airplane mode 283

AlertDialog class 118, 174

AlertDialog.Builder class 118

 setItems method 297, 313

alpha (transparency) values 57

alpha animation for a View 128

alpha method of class Color 209

alternative-resource naming
 conventions 68

Amazon Mobile app 392

analysis 20

Android 2.2 (Froyo) 7

Android 2.3 (Gingerbread) 7

Android 3.x

 Honeycomb 8

Android 4.0 (Ice Cream

 Sandwich) 8

Android 6.0

 permissions 167, 176, 195

Android 6.0 (Marshmallow)

 permissions 176, 181

Android APIs 4

Android app marketplaces 398

 Amazon Appstore 398

 AndroidPIT 398

 Appitalism 398

 GetJar 398

 Moboro 398

 Opera Mobile Store 398

 SlideMe 398

Android Beam 9, 10

- Android Cloud to Device Messaging (C2DM) 7
- Android Design Support Library xxiv, 121, 122, 264, 265, 298
- Android developer documentation (`developer.android.com`) xxvi
- Android Developers Blog 388
- Android device manufacturers xxi
- Android Device Monitor 389
- Android device type for a project 40
- Android emulator **xxxiii**, **16**, 36
 - Android for Programmers* website xxi, xxvi
- Android Jelly Bean **9**
- Android KitKat **10**
- Android Lint 60
- Android Lollipop 11
- Android Market
 - language 396
 - location 397
 - price 397
- Android Marshmallow **12**
- Android Newsgroups
 - Android Discuss 33
- Android project
 - res folder **46**, **54**
 - values folder **55**
- Android SDK xxi, xxii, xxxi, xxxiii, **2**, **16**
 - versions and API levels 42
- Android services
 - access 142
- Android Software Development Kit (SDK) xxxii
- Android source code and documentation
 - FAQs 4
 - licenses 4
 - source code 3, 4
- Android Studio xxxi, xxxii, **2**, **16**, **36**, **37**
 - Component Tree window 74, 78
 - documentation 33
- Android Studio (cont.)
 - layout editor 36, 37, 38, 44, 46, 47, 49, 52, 60, 67
 - Screen Record tool 390
 - Tips and Tricks xxxii
- Android Support Library xxiv, 77, 78, 114, 176, 177, 206
- Android TV xxv
- Android versions
 - Android 1.5 (Cupcake) 7
 - Android 1.6 (Donut) 7
 - Android 2.0–2.1 (Eclair) 7
 - Android 2.2 (Froyo) 7
 - Android 2.3 (Gingerbread) 7
 - Android 3.0–3.2 7
 - Android 4.0 (Ice Cream Sandwich) 7
 - Android 4.1–4.3 7
 - Android 4.4 7
 - Android 5 7
 - Android 6 7
- Android Virtual Device (AVD) **xxxiv**, **16**
 - Setting hardware emulation options 30
- Android Virtual Device Manager **24**
- Android Virtual Devices (AVDs) xxxiii, 24
- Android Wear xxv
- android XML namespace 179
- android:allowBackup attribute of the manifest application element **101**
- android:background attribute of a TextView 335
- android:colorAccent 89
- android:colorPrimary 89
- android:colorPrimaryDark 89
- android:duration attribute of a translate animation **129**
- android:fromXDelta attribute of a translate animation **129**
- android:icon attribute of the manifest application element **101**
- android:id attribute **54**
- android:label attribute of the manifest activity element **102**
- android:label attribute of the manifest application element **101**
- android:layout_gravity attribute 60
- android:name attribute of the manifest activity element **102**
- android:screenOrientation attribute of the manifest activity element **102**
- android:startOffset attribute of a translate animation **129**
- android:supportsRtl attribute of the manifest application element **101**
- android:theme attribute of the manifest application element **101**
- android:toXDelta attribute of a translate animation **129**
- android:windowSoftInputMode attribute of the manifest activity element **102**
- android.app package **76**, 113, 114, 118, 330
- android.content package **115**, 142, 175, 330, 343
- android.content.res package **115**, **117**, **143**
- android.database package **329**
- android.database.sqlite package **329**, 348
- android.graphics package 175
- android.graphics.drawable package 155
- android.intent.action.MAIN 103
- android.intent.category.LAUNCHER 103
- android.media package 220, 221
- android.net package **311**
- android.os package 93, 117, **263**

- android.permission.WRITE_EXTERNAL_PERMISSION 176
- android.preference package 113
- android.provider package 329, 344
- android.support.design.widget package 265
- android.support.v4.app package 113, 114
- android.support.v7.app package 77, 93
- android.support.v7.widget package 296
- android.text package 79, 93
- android.util package 119, 239
- android.view package 113, 220
- android.view.animation package 117
- android.view.inputmethod package 280
- android.widget package 78, 81, 94, 117, 263, 264
- Android@Home framework 9
- AndroidLicenser 398
- AndroidManifest.xml 81, 120, 121
 - action element 103
 - activity element 102
 - application element 101
 - category element 103
 - intent-filter element 103
 - manifest element 101
 - provider element 332
- anim folder of an Android project 46, 115, 116
- animation xxv, 110
 - alpha animation for a View 128
 - framework 8
 - manual 221
 - options in an XML file 117
 - rotate animation for a View 128
 - scale animation for a View 128
 - set 128
 - thread 221
- animation (cont.)
 - translate animation for a View 128
 - tween 128
 - View based 128
- animation circular reveal 156
- Animation class 117
 - setRepeatCount method 117, 151
- AnimationUtils class 117, 151
 - loadAnimation method 117, 151
- Animator class
 - circular reveal 118
 - setDuration method 118
 - start method 118
- animator folder of an Android project 46, ■
- AnimatorListenerAdapter class 157
 - onAnimationEnd method 157
- anonymous inner class 76, 79
- ANR (activity not responding) dialog 263, 330
- ANR (Application Not Responding) dialog 95, 296
- anti-aliasing 199
- API key (web services) 260
- .apk file (Android application package file) 388
- app xxxi
- app bar 43, 77, 107
- app development xxxi
- app icon
 - adding 51
- app linking 13
- app platforms
 - Android 399
 - iPhone 399
 - Windows 399
- app resources 38, 46
- app review sites
 - Android and Me 401
 - Android App Review Source 401
 - Android Police 401
 - AndroidGuys 401
 - AndroidLib 401
 - AndroidPIT 401
- app review sites (cont.)
 - Androinica 401
 - AppBrain 401
 - Applicicious 401
 - Appstorm 401
 - AppZoom 401
 - Best Android Apps Review 401
 - Phandroid 401
- app review video sites
 - Applicicious 401
 - Crazy Mike's Apps 401
 - Life of Android 401
 - State of Tech 401
- app templates xxiii
- app XML namespace 179
- AppCompatActivity class 77, 93, 94
- app-driven approach xxii, 2
- appendWhere method of a SQLiteQueryBuilder 349
- application element in AndroidManifest.xml 101
- Application Not Responding (ANR) dialog 95, 296
- application resource 15
- apply method of class
 - SharedPreferences.Editor 146, 310
- ARGB 207
- ARGB color scheme 169
- argb method of class Color 210
- ARGB_8888 constant 200
- ArrayAdapter class 263, 272
 - getView method 273
- ArrayList class 120, 263
- ART runtime 12
- asset 395
- AssetManager class 115
 - list method 152
- assets folder of an Android app 115
- AsyncTask class 263, 276, 276, 277, 282, 282, 330
 - execute method 275
- AsyncTaskLoader class 330
- attribute
 - in the UML 20
 - of a class 18
 - of an object 20

AttributeSet class 239
 audio xxv, 15
 audio stream
 music 226
 audio streams 220
 music 221
 audio volume 221
AudioAttributes class 221
 setUsage method 221, 240
AudioAttributes.Builder
 class 240
 setAudioAttributes
 method 240
AudioManager class 221, 226
 authority (ContentProvider)
 342
 automatic backup 13
AVD (Android Virtual Device)
 xxxiv, 16

B

back stack 328, 356, 357, 358,
 359
 pop 356, 357
 push 357
 background
 activity sent to 226
 background property of a view
 87, 88
 base Uri of a ContentProvider
 342
BaseColumns interface 344
 beginTransaction method of
 class **FragmentManager** 356
 behavior
 of a class 18
 Bezier curve 204
 bind data to a **ListView** 263
Bitmap class 175, 214
 bitmap encoding 200
 createBitmap method 200
 eraseColor method 214
 recycle method 200
Bitmap.Config.ARGB_8888
 constant 200
BitmapFactory class 277
 decodeStream method 277
Blank Activity template 43
blue method of class **Color** 209
Bluetooth Health Devices 9

bluetooth stylus support 13
 brand awareness 392
 branding apps
 Amazon Mobile 392
 Bank of America 392
 Best Buy 392
 CNN 392
 Epicurious Recipe 392
 ESPN ScoreCenter 392
 NFL Mobile 392
 NYTimes 392
 Pocket Agent 392
 Progressive Insurance 392
 UPS Mobile 392
 USA Today 392
 Wells Fargo Mobile 392
 Women's Health Workouts
 Lite 392
 broadcast receiver 76
Build.VERSION.SDK_INT 253
Bundle class 93, 96
 for an Intent 314
 putParcelable method 357
Button class
 lines property 137
 textColor property 137

C

C2DM (Android Cloud to
 Device Messaging) 7
Calendar API 9
 callback methods 329
 camera 5
Cannon Game app 17
Canvas class 175, 175
 drawBitmap method 201
 drawCircle method 236
 drawLine method 233
 drawPath method 201, 205
 drawRect method 247
 drawText method 248
 carrier billing 391
 case-insensitive sort 307
 category element of the
 manifest file 103
 cell in a **GridLayout** 81
 characteristics of great apps 31
 check-in 400
 circular reveal animation 156
 circular reveal **Animator** 118

class 15, 19
 instance variable 20
 class library 4
 classes
 Activity 76, 77, 93
 ActivityNotFoundException
 on 119, 296
 Adapter 263
 AdapterView 263
 AlertDialog 118
 AlertDialog.Builder 118
 Animation 117
 AnimationUtils 117, 151
 AnimatorListenerAdapter
 157
 AppCompatActivity 77, 93
 ArrayAdapter 263, 272
 ArrayList 120, 263
 AssetManager 115
 AsyncTask 263, 330
 AsyncTaskLoader 330
 AttributeSet 239
 AudioManager 221, 226
 Bitmap 175, 214
 BitmapFactory 277
 Bundle 93, 96
 Canvas 175, 175
 Collections 120
 Color 209
 Configuration 117, 143
 ContentProvider 329, 346
 ContentResolver 175, 330,
 342, 347, 350, 351, 353,
 354, 374, 380
 ContentUris 343
 ContentValues 350, 374
 Context 142
 CoordinatorLayout 121
 Cursor 329
 CursorFactory 345
 CursorLoader 330
 DialogFragment 114, 159,
 160
 DialogInterface 119
 Drawable 155
 EditText 78, 94, 265
 FloatingActionButton
 264
 Fragment 113
 FragmentManager 114, 328

classes (cont.)

FragmentTransaction **114**,
 328, **356**, **357**
 FrameLayout **224**
 GridLayout **78**, **81**
 Handler **117**
 HttpURLConnection **277**
 ImageView **38**, **60**
 InputMethodManager **280**
 InputStream **155**
 Intent **102**, **119**
 JSONArray **263**
 JSONObject **262**
 LayoutInflater **114**
 LinearLayout **37**, **48**
 LinearLayoutManager **297**
 ListPreference **115**
 Loader **330**, **363**
 LoaderManager **330**, **360**,
 363
 Log **119**, **152**
 MediaStore **175**
 MediaStore.Images.Media
 175
 Menu **113**, **144**
 MenuInflater **144**
 MotionEvent **175**, **202**, **220**,
 251
 MultiSelectListPreference
 115
 NumberFormat **78**, **94**, **270**
 Paint **175**
 Path **175**
 Preference **115**
 PreferenceFragment **113**,
 115, **161**
 PreferenceManager **115**,
 141, **142**
 PrintHelper **206**
 R **96**
 R.drawable **96**
 R.id **96**
 R.layout **96**
 R.string **96**
 RecyclerView **296**, **307**
 RecyclerView.Adapter
 297, **307**, **332**
 RecyclerView.ItemDecorati
 on **297**, **307**, **332**

classes (cont.)

RecyclerView.LayoutMana
 ger **297**
 RecyclerView.ViewHolder
 297
 RelativeLayout **48**
 Resources **143**
 ScrollView **338**
 SeekBar **74**, **78**, **94**
 Sensor **174**
 SensorEvent **193**
 SensorManager **190**
 SharedPreferences **115**,
 295, **305**
 SharedPreferences. Edito
 r **115**, **146**, **296**, **309**, **310**
 Snackbar **265**, **267**, **280**
 SoundPool **220**, **240**
 SQLiteDatabase **329**
 SQLiteOpenHelper **329**
 SQLiteQueryBuilder **348**
 SurfaceHolder **221**, **240**
 SurfaceView **221**, **240**
 TextView **38**, **54**, **78**, **94**
 Thread **221**, **252**
 Toast **117**, **147**
 Uri **311**, **330**
 UriMatcher **346**
 View **76**, **221**
 ViewAnimationUtils **118**
 ViewGroup **338**
 client area **37**
 cloud computing **7**
 Cloud Test Lab xxv, xxxiv, **386**
 code file **395**
 code highlighting xxv, **2**
 code license xxi
 code walkthrough **2**
 code-completion **46**
 code-completion window **53**
 code-folding xxxiii
 collection
 shuffle **155**
 Collections class **120**
 shuffle method **120**
 sort method **307**
 collision detection **221**, **235**, **244**
 color **175**
 hue **80**
 shade **80**

Color class **209**

 alpha method **209**
 argb method **210**
 blue method **209**
 green method **209**
 red method **209**
 color folder of an Android
 project **46**, **115**, **116**
 color state list **118**
 color state list resource **126**
 color state list resource file **118**
 colorAccent **89**
 colorPrimary **89**
 colorPrimaryDark **89**
 colors.xml **126**
 columnCount property of a
 GridLayout **83**
 commit method of class
 FragmentTransaction **356**
 commit method of class
 SharedPreferences.Editor
 147
 company domain name used in
 a package **40**
 Comparator<String> object
 String.CASE_INSENSITIVE
 _ORDER **307**
 compiling apps **386**
 component **18**
 Component Tree window in
 Android Studio **45**, **74**, **78**,
 83, **136**
 concurrent documents and
 activities **12**
 Configuration class **117**, **143**,
 144
 orientation instance
 variable **117**
 Constants
 MODE_PRIVATE **306**
 MODE_WORLD_READABLE **306**
 MODE_WORLD_WRITABLE **306**
 content provider **76**
 contentDescription property
 67
 contentDescription property
 of a View **135**
 ContentProvider
 base Uri **342**

- ContentProvider authority 342
 - ContentProvider class 329, 346
 - delete method 353
 - getType method 348
 - insert method 350
 - onCreate method 347
 - query method 348
 - update method 352
 - ContentResolver class 175, 330, 342, 347, 350, 351, 353, 354, 374, 380
 - delete method 380
 - insert method 374
 - update method 374
 - ContentUri class 343
 - ContentValues class 350, 374
 - Context class 142
 - getSharedPreferences method 306
 - getSystemService method 280
 - startActivity method 311
 - ContextWrapper class
 - getAssets method 152, 155
 - control 17
 - coordinating efforts between separate apps 102
 - CoordinatorLayout class 121
 - corners element of a shape 335
 - crash report 398
 - create a new layout 139
 - Create New Project dialog 39, 82, 83, 121, 176, 222, 265, 298, 331
 - createBitmap method of class Bitmap 200
 - createChooser method of class Intent 314
 - createCircularReveal method of class ViewAnimationUtils 118, 157
 - createFromStream method of class Drawable 155
 - creating a dimension resource 56
 - creative commons public license 260
 - cryptographic key 385
 - CT
 - Google Play and App Business Issues 384
 - Cursor class 329
 - getColumnIndex method 368
 - getLong method 368
 - getString method 368
 - moveToFirst method 375, 381
 - moveToPosition method 368
 - setNotificationUri method 350
 - CursorFactory class 345
 - CursorLoader class 330
 - custom subclass of View 237
 - custom view 174
 - customize the keyboard 339
- D**
- dark keyboard 74
 - data binding 263
 - Data Binding support library 97
 - database version number 345
 - Daydream 10
 - DDMS perspective
 - LogCat tab 119
 - debugging
 - logging exceptions 119, 152
 - decodeStream method of class BitmapFactory 277
 - default preferences 141
 - default resources 68
 - define a new style 334
 - Deitel Buzz Online Newsletter 403
 - Deitel Facebook page 400
 - Deitel Training 404
 - delete method of a
 - ContentProvider 353
 - ContentResolver 380
 - delete method of class SQLiteDatabase 354
 - density-independent pixels
 - dp 56, 56
 - dependencies
 - adding to project 298
 - design preview in layout XML editor 46
 - design process 20
 - Design tab in the layout editor 37
 - developer documentation
 - Core App Quality 386
 - Keeping Your App Responsive 33
 - Launch Checklist 386
 - Localization Checklist 386
 - Performance Tips 33
 - Signing Your Applications 388
 - Tablet App Quality 386
 - developer options 10
 - developer registration 394
 - device configuration 15
 - Device Screen Capture window 389
 - dialog
 - negative action 118
 - neutral action 118
 - positive action 118
 - DialogFragment class 114, 159, 160
 - onCreateDialog method 159
 - show method 159
 - DialogInterface class 119
 - DialogInterface.OnClickListener
 - tener interface 119, 160
 - digital certificate 388
 - digitally sign your app 388
 - digits property of an EditText 87
 - @dimen resource 56
 - dimens.xml 134
 - dimension resource 56, 134
 - creating 56
 - Direct Share 13
 - disabilities 38, 66
 - disconnect method of class HttpURLConnection 277
 - divider property of a
 - LinearLayout 336
 - documentation
 - Android Design 33
 - App Components 32

- documentation (cont.)
 - Class Index* 32
 - Data Backup* 33
 - Debugging* 33
 - Getting Started with Publishing* 33
 - Google Play Developer Distribution Agreement* 33
 - Launch Checklist (for Google Play)* 33
 - Managing Your App's Memory* 33
 - Package Index* 32
 - Security Tips* 33
 - Tools Help* 33
 - Using the Android Emulator* 32
- doInBackground method of class AsyncTask 276, 282, 282
- domain name used in a package 40
- downloading source code xxvi
- dp (density-independent pixels) 56
- drag event 204
- draw
 - circles 175
 - lines 175
 - text 175
- Drawable class 155
 - createFromStream method 155
- drawable folder of an Android project 47
- Drawable resource
 - shape element 335
- drawBitmap method of class Canvas 201
- drawCircle method of class Canvas 236
- drawing characteristics 175
 - color 175
 - font size 175
 - line thickness 175
- drawLine method of class Canvas 233
- drawPath method of class Canvas 201, 205
- drawRect method of class Canvas 247
- drawText method of class Canvas 248
- drive sales 392
- E**
 - e method of class Log 152
 - edit method of class SharedPreferences 146, 309
 - Editable interface 93
 - EditText
 - imeOptions 339, 340
 - inputType 339, 340
 - EditText class 78, 94, 265
 - digits property 87
 - input type 84
 - maxLength property 87
 - restrict maximum number of digits 78
 - elevation 122
 - elevation property of a view 80, 87, 88
 - emulator 16, 386
 - gestures 17
 - emulator functionality 17
 - emulator gestures and controls 17
 - encapsulation 20, 20
 - End User License Agreement (EULA) 386
 - eraseColor method of class Bitmap 214
 - event handling 76
 - events 4
 - execSQL method of class SQLiteDatabase 345
 - executable components
 - activity 76
 - broadcast receiver 76
 - content provider 76
 - service 76
 - execute method of class AsyncTask 275
 - explicit Intent 119, 145, 296
 - Explore by Touch 36, 38, 66
 - Extensible Markup Language (XML) 38
 - externalizing resources 54
- F**
 - FAB (floating action button) 264
 - Fabric (Twitter's mobile development platform) 320
 - face detection 9
 - Facebook 293, 400
 - Deitel 400
 - Deitel page 400
 - featured image 388
 - final local variable for use in an anonymous inner class 312
 - financial transaction 394
 - findFragmentById method of class Activity 114, 143
 - floating action button (FAB) 264
 - FloatingActionButton class 122, 264
 - hide method 308
 - show method 308
 - folders
 - assets 115
 - res/drawable 335
 - res/raw 220, 223
 - font size 175
 - format method of class NumberFormat 97
 - format specifier
 - multiple in a String resource 124
 - numbering in a String resource 124
 - forums 33
 - Android Forums 33
 - Stack Overflow 33
 - fragment 8, 113
 - Fragment class 76, 113
 - getActivity method 151
 - getLoaderManager method 363
 - getString method 146, 151
 - onActivityCreated method 226, 363
 - onAttach method 174, 209, 363, 370, 378
 - onCreate method 114
 - onCreateOptionsMenu method 114, 193

- Fragment class (cont.)
 - onCreateView method 114, 149, 226
 - onDestroy method 220, 227
 - onDetach method 174, 209, 363, 370, 378
 - onOptionsItemSelected method 114, 193
 - onPause lifecycle method 191
 - onPause method 173, 226
 - onRequestPermissionsResult method 196
 - onResume lifecycle method 190
 - requestPermissions method 195, 196
 - setArguments method 357
 - setHasOptionsMenu method 189
 - shouldShowRequestPermissionRationale method 195
- Fragment layout 132
- Fragment lifecycle 174, 363, 370, 378
- fragment lifecycle 114
- Fragment lifecycle methods 209
- FragmentActivity class
 - getSupportFragmentManager method 114, 143, 145
- FragmentManager class 114, 328
 - beginTransaction method 356
 - getFragmentByTag method 159
 - popBackStack method 356
- FragmentTransaction class 114, 328, 356, 357
 - add method 356
 - addToBackStack method 357
 - commit method 356
 - replace method 357
- FrameLayout class 224
- free app 390
- Froyo (Android 2.2) 7
- Fullscreen Activity template 43
- fully qualify a custom View's class name in an XML layout 174
- future proof 32
- G**
- game loop 221, 244, 253
- games 31
- gaming console 5
- gesture 5
 - double tap 5
 - double touch 5
 - drag 5
 - long press 5
 - pinch 5
 - pinch zoom 5
 - Swipe 5
 - tap 5
 - touch 5
- getActionIndex method of class MotionEvent 203
- getActionMasked method of class MotionEvent 202
- getActivity method of class Fragment 151
- getAll method of class SharedPreferences 307
- getAssets method of class ContextWrapper 152, 155
- getColumnIndex method of class Cursor 368
- getConfiguration method of class Resources 143, 144
- getDefaultSensor method of class SensorManager 190
- getDouble method of class JSONObject 284
- getFragmentByTag method of class FragmentManager 159
- getFragmentManager method of class Activity 114, 159
- getHolder method of class SurfaceView 240
- getItemCount method of class RecyclerView.Adapter 318, 368
- getItemID method of class MenuItem 193
- getJSONArray method of class JSONObject 284
- getJSONObject method of class JSONArray 284
- getLastPathSegment method of class Uri 349
- getLoaderManager method of class Fragment 363
- getLong method of class Cursor 368
- getLong method of class JSONObject 284
- getMenuInflater method of class Activity 144
- getPointerCount method of class MotionEvent 204
- getReadableDatabase method of class SQLiteOpenHelper 350
- getResources method of class Activity 143, 144
- getSharedPreferences method of class Context 306
- getString method of class Activity 313
- getString method of class Cursor 368
- getString method of class Fragment 146, 151
- getString method of class JSONObject 284
- getString method of class SharedPreferences 311
- getStringSet method of class SharedPreferences 146
- getSupportFragmentManager method of class FragmentActivity 114, 143, 145
- getSystemService method of class Context 280
- getSystemService method of classd Activity 190
- getTag method of class View 264
- getType method of a ContentProvider 348
- getView method of class ArrayAdapter 273
- getWritableDatabase method of class SQLiteOpenHelper 351

getX method of class
 MotionEvent 204
 getY method of class
 MotionEvent 204
Go to next state button 64
 Google APIs 4
 Google Cloud Messaging 7
 Google Maps 5
 Google Payments 385, **395**
 Google Payments merchant
 account 391, 393
 Google Play **13**, 385, 391, 394
 countries 397
 crash report 398
 high-resolution app icon 395
 promotional graphic 395
 promotional video 390, 396
 publish 395
 Publish an Android App on
 Google Play 395
 publisher account 393
 screenshots 395
 Google Play Developer Console
 398
Google Play Developer Program
 Policies 394
 Google Wallet 391
 Google+ 293
 Gradle build system **176**
Graphical Layout editor in the
 Android Developer Tools **38**
 graphics xxv, 15
 graphics processing unit (GPU)
 xxxiii
 gravity property of a TextView
 58, 88
 gravity sensor 174
 green guide lines in layout editor
 60
 green method of class Color
 209
 GridLayout
 columnCount property 83
 layout:column of a view 84
 layout:columnSpan of a
 view 84
 layout:row of a view 84
 rowCount property 83
 useDefaultMargins
 property **83**

GridLayout class **78**, 81
 gesture 17
 GUI
 layout 46
 view (component) 46
 GUI component
 view 37
 GUI components
 EditText **78**
 ImageView **38**, 49, 60
 naming convention 82
 programmatically create 114
 ScrollView **338**
 SeekBar **74**, **78**
 TextView **38**, 49, 54
 ViewGroup **338**
 GUI design 31
 screen type 47
 GUI thread 117, 263, 330
 guide lines in layout editor 60
 gyroscope sensor 174

H

Handler class **117**
 postDelayed method **117**,
 160
 hardware accelerated execution
 manager (HAXM) xxxiv
 hardware support 15
 hashtag 400
 height of a table row 81
 hide method of class
 FloatingActionButton **308**
 hide the soft keyboard 308
 hideSoftInputFromWindow
 method of class
 InputMethodManager **280**
 hierarchical parent of an
 Activity 163
 hint property of a TextView **86**
 Holo user interface 8
 host a Fragment in an Activity
 114
 HttpURLConnection class 277
 disconnect method 277
 openConnection method
 277
 hues of a color 80
 HyperText Transfer Protocol
 (HTTP) 260

I

i-Newsire 402
 icon 386, **387**
 icon design firms
 99designs 387
 Elance 387
 glyphlab 387
 Iconiza 387
 id property of a layout or
 component **53**
 images xxv
 ImageView **38**, 49, 60
 ImageView class
 adjustViewBounds property
 135
 scaleType property 135
 src property **53**, 62
 imeOptions of an EditText
 339, 340
 immersive mode 222, 247, 253,
 254
 implicit Intent **119**, **296**, 311
 in-app advertising 390, **392**
 in-app billing 393
 security best practices 393
 in-app purchase 390, 394
 <include> element in a layout
 XML file 122
 inflate method of class
 LayoutInflater **149**
 inflate method of class
 MenuInflater **144**
 inflate the GUI 241
 inflating a GUI **96**
 information hiding **20**
 inheritance **20**
 in-memory database 345
 input type of an EditText 84
 InputMethodManager class **280**
 hideSoftInputFromWindow
 method **280**
 InputStream class 155
 setImageDrawable method
 155
 inputType of an EditText 339,
 340
 insert method of a
 ContentProvider **350**
 insert method of a
 ContentResolver 374

insert method of class
 SQLiteDatabase 351

insertImage method of class
 MediaStore.Images.Media
 175

instance 19

instance variable 20

Intel HAXM emulator xxxiv

IntelliJ® IDEA xxxii

intent chooser 293, 314

Intent class 102, 119

- ACTION_SEND constant 313
- ACTION_VIEW constant 311
- Bundle 314
- createChooser method 314
- explicit 119, 296
- implicit 119, 296
- putExtra method 314
- resolveActivity method 119

intent extras 314

intent filter 119

intent messaging 102, 119

intent-filter element in
 AndroidManifest.xml 103

intents

- coordinating efforts between
 separate apps 102
- launching activities 103

interfaces

- BaseColumns 344
- DialogInterface.OnClickListener 119, 160
- Editable 93
- implementing methods in
 Java 99
- List 120
- LoaderManager.LoaderCallback 330
- OnClickListener 311
- OnSeekBarChangeListener 98
- Runnable 117
- SeekBar.OnSeekBarChangeListener 79, 94, 210
- SensorEventListener 192
- Set 120
- SurfaceHolder.Callback 221, 240, 250
- TextWatcher 79, 93, 307

interfaces (cont.)

- View.OnClickListener 151
- View.OnLongClickListener 311

internationalization 37, 38, 67, 78

Internet public relations
 resources

- ClickPress 402
- eReleases 402
- Marketwired 402
- Mobility PR 402
- Newswire 402
- openPR 402
- PR Leap 402
- PRLog 402
- PRWeb 402

invalidate method of class
 View 200

invoke a REST web service 282

J

Java xxii, 4

Java for Programmers xxii

Java Fundamentals xxii

Java How to Program xxii

Java SE 7 Software
 Development Kit xxxi

java.io package 155

java.text package 78, 94

java.util package 120

JavaScript Object Notation
 (JSON) 261

JDK 7 xxxi

JetBrains xxxii

join operations 349

JSON (JavaScript Object
 Notation) 257, 261

JSONArray class 263

- getJSONObject method 284
- length method 284

JSONObject class 262

- getDouble method 284
- getJSONArray method 284
- getLong method 284
- getString method 284

K

keyboard 5

keyboard types 339

keySet method of interface Map
 307

key-value pairs associated with
 an app 114, 295

L

landscape mode 241

landscape orientation 102

landscapeorientation 36

large-screen device 8

launch another app 311

launching activities 103

launchMode of the <activity>
 element 163

- "singleTop" 163
- "standard" 163

layout 16, 37

- create new 139
- layout (GUI) 46
- layout editor 36, 37, 38, 44, 46, 47, 49, 52, 60, 67
- Design tab 37, 46
- guide lines 60
- Palette 48
- Text tab 37, 46
- tooltip 61

layout folder of a project 47

layout XML editor

- design preview 46

layout:column of a view in a
 GridLayout 84

layout:columnPan of a view in
 a GridLayout 84

layout:gravity property of a
 view 59, 61, 86, 87, 88, 134

layout:margin property of a
 view 134

layout:row of a view in a
 GridLayout 84

layout:weight property of a
 view 60, 61, 134

LayoutInflater class 114

- inflate method 149

layouts

- activity_main.xml 49
- GridLayout 78, 81
- LinearLayout 37, 48
- RelativeLayout 48

length method of class
 JSONArray 284

- license for Android 4
 - licensing policy 388
 - licensing service **388**
 - lifecycle methods 173, 220
 - lifecycle methods of an app 93
 - light sensor 174
 - line thickness 175
 - linear acceleration sensor 174
 - LinearLayout 37, 48
 - orientation property 53
 - LinearLayoutManager class (for RecyclerViews) 297, 307
 - lines property of a Button 137
 - linking your apps 397
 - Lint, Android 60
 - Linux 16
 - List interface **120**
 - list method of class
 - AssetManager 152
 - ListPreference class **115**
 - ListView class 360
 - data binding 263
 - performance 264
 - setAdapter method 279
 - smoothScrollToPosition method 283
 - load method of class SoundPool 241
 - loadAnimation method of class AnimationUtils 117, 151
 - Loader class 330, 363
 - LoaderManager class 330, 360, 363
 - LoaderManager.LoaderCallbacks interface 330
 - onCreateLoader method 365, 374, 381
 - onLoaderReset method 365, 375, 381
 - onLoadFinished method 365, 374, 381
 - localization 37, 38, 54, 67, 124
 - Localization Checklist 71
 - localized resources 68
 - lock screen widgets 10
 - lockCanvas method of class SurfaceHolder 253
 - Log class 119, 152
 - e method 152
 - LogCat in Android Studio 119
 - LogCat tab in the Android
 - DDMS perspective 119
 - logcat tool **119**
 - logging 119
 - logging exceptions 119, 152
 - long press 291
 - long-running operations 263, 330
- M**
- Mac OS X 16
 - magnetic field sensor 174
 - main thread 117
 - makeText method of class Toast 147
 - manifest activity element
 - android:label attribute **102**
 - android:name attribute **102**
 - android:screenOrientation attribute **102**
 - android:windowSoftInputMode attribute **102**
 - manifest application element
 - android:allowBackup attribute **101**
 - android:icon attribute **101**
 - android:label attribute **101**
 - android:supportsRtl attribute **101**
 - android:theme attribute **101**
 - manifest element in AndroidManifest.xml 101
 - manifest file 395
 - manually perform an animation 221
 - Map interface
 - keySet method **307**
 - mashup 5
 - Master/Detail Flow template 43
 - material design 11, 12, 264
 - color palette 38, 57
 - icons 127
 - Material themes xxiii
 - specification xxiii
 - vector icons 178
 - max property of a SeekBar 88
 - maxLength property of an
 - EditText 87
 - media files 220
 - MediaStore class 175
 - MediaStore.Images.Media class 175
 - insertImage method 175
 - medium sized font 84
 - Menu class 113, 144, 193
 - menu folder of an Android project 46, 115, 116
 - menu item
 - showAsAction 128, 179
 - MenuInflater class 144, 193
 - inflate method 144
 - MenuItem class
 - getItemID method 193
 - merchant account 391, 395
 - method **19**
 - method call **19**
 - micro blogging 399, 400
 - MIME type 314
 - minimum screen width qualifier 116
 - mipmap 51
 - mipmap folder of an Android project 46
 - mipmap resource folder 51
 - mobile advertising 391
 - mobile advertising network 392
 - AdMob 392
 - mobile advertising networks 402
 - AdMob 403
 - InMobi 403
 - Medialets 403
 - Millennial Media 403
 - mMedia 403
 - Smaato 403
 - Tapjoy 403
 - mobile payment provider 394
 - Boku 394
 - PayPal Mobile Libraries 394
 - Samsung In-App Purchase 394
 - mobile payment providers 394
 - modal dialog **118**
 - MODE_PRIVATE constant **306**
 - MODE_WORLD_READABLE constant **306**

MODE_WORLD_WRITABLE constant
306

monetizing apps 385, 392

MotionEvent class 175, 202,
220, 251

- getActionIndex method
203
- getActionMasked method
202
- getPointerCount method
204
- getX method 204
- getY method 204

moveTo method of class Path
203

moveToFirst method of class
Cursor 375, 381

moveToPosition method of
class Cursor 368

MP3 player 5

multimedia xxv

multiple format specifiers 124

MultiSelectListPreference
class 115

multitouch 202

multitouch screen 5

music audio stream 221, 226

N

naming convention

- GUI components 82

near-field communication
(NFC) 8

negative action in a dialog 118

network access 15

neutral action in a dialog 118

New String Value Resource dialog
55, 86

newsgroups 33

- Android Developers 33

newsletter

- Deitel Buzz 403

notifyDataSetChanged
method 263, 283, 310

notifyDataSetChanged
method of class
ArrayAdapter 263, 283, 310

NumberFormat class 78, 94, 270

- format method 97

numbering format specifiers
124

numeric input 78

numeric keypad 74

O

obfuscate code 388

object 18

object (or instance) 20

object-oriented analysis and
design (OOAD) 20

object-oriented language 20

object-oriented programming
(OOP) 21

Oceania 109

OEM original equipment
manufacturer 4

offset method of class Rect
236

onActivityCreated method of
class Fragment 226, 363

onAnimationEnd method of
class
AnimatorListenerAdapter
157

onAttach method of class
Fragment 174, 209, 363,
370, 378

onBindViewHolder method of
class RecyclerView.Adapter
318, 368

onCreate method of a
ContentProvider 347

onCreate method of class
Activity 77, 95, 173

onCreate method of class
Fragment 114

onCreate method of class
SQLiteOpenHelper 345

onCreateDialog method of
class DialogFragment 159

onCreateLoader method of
interface
LoaderManager.Loader-
Callbacks 365, 374, 381

onCreateOptionsMenu method
of class Activity 113, 144

onCreateOptionsMenu method
of class Fragment 114, 193,
379

onCreateView method of class
Fragment 114, 149, 226

onCreateViewHolder method
of class
RecyclerView.Adapter 318,
368

onDestroy method of class
Activity 173, 220

onDestroy method of class
Fragment 220, 227

onDetach method of class
Fragment 174, 209, 363,
370, 378

onDowngrade method of class
SQLiteOpenHelper 345

onDraw method of class View
201

onLoaderReset method of
interface LoaderManager.
LoaderCallbacks 365, 375,
381

onLoadFinished method of
interface LoaderManager.
LoaderCallbacks 365, 374,
381

onLongClick method of
interface OnLongClick-
Listener 311

OnLongClickListener interface
onLongClick method 311

OnLongClickListener nested
interface of class View 311

onOptionsItemSelected
method of class Activity
113, 145

onOptionsItemSelected
method of class Fragment
114, 193, 379

onPause method of class
Activity 173, 173

onPause method of class
Fragment 173, 191, 226

onPostExecute method 277,
282

onPostExecute method of class
AsyncTask 277, 282

onProgressChanged method of
interface SeekBar.OnSeek-
BarChangeListener 79

onProgressUpdate method of class AsyncTask 277, 282
 onRequestPermissionsResult method of class Fragment 196
 onResume method of class Activity 173, 173
 onResume method of class Fragment 190
 OnSeekBarChangeListener interface 98
 onSensorChanged method 192
 onSensorChanged method of interface SensorEventListener 192
 onSizeChanged method of class View 200, 241
 onStart method of class Activity 143, 173
 onStop method of class Activity 173
 onTextChanged method of interface TextWatcher 79
 onTouchEvent method of class View 202
 onTouchEvent method of class View 175, 220, 251
 onUpgrade method of class SQLiteOpenHelper 345
 OOAD (object-oriented analysis and design) 20
 OOP (object-oriented programming) 21
 Open Handset Alliance 6
 open source 3
 open source apps 4
 Open Source Project discussion groups 3
 openConnection method of class HttpURLConnection 277
 openPR 402
 operating system 6
 operating system requirements xxxi
 operating systems services 15
 options menu 27, 113, 167, 168
 orange guide lines in layout editor 60
 org.json package 262

orientation
 landscape 36
 portrait 36
 orientation instance variable of class Configuration 117
 orientation property of a LinearLayout 53
 orientation qualifier 116
 orientation sensor 174
 ORIENTATION_LANDSCAPE constant 117
 ORIENTATION_PORTRAIT constant 117
 original equipment manufacturer (OEM) 4
 overflow options menu 167, 168

P

package 14
 package name 40
 packages
 android.animation 15
 android.app 15, 76, 113, 114, 118, 330
 android.content 15, 115, 142, 175, 330, 343
 android.content.res 15, 115, 117, 143
 android.database 15, 329
 android.database.sqlite 15, 329, 348
 android.graphics 15, 175
 android.graphics.drawable 15, 155
 android.hardware 15
 android.media 15, 220, 221
 android.net 15, 311
 android.os 15, 93, 117
 android.preference 15, 113
 android.provider 15, 329, 344
 android.support.design.widget 265
 android.support.v4.app 113, 114
 android.support.v7.app 77, 93

packages (cont.)
 android.support.v7.widget 296
 android.text 16, 79, 93
 android.util 16, 119, 239
 android.view 16, 113, 220
 android.view.animation 117
 android.view.inputmethod 280
 android.widget 16, 78, 81, 94, 117
 java.io 155
 java.util 120
 java.text 78, 94
 org.json 262
 padding element of a shape 335
 padding property of a view 87, 87, 88
 paid app
 average price 391
 Paint class 175
 filled shape with a border 199
 filled shape without a border 199
 line 199
 setAntiAlias method 199
 setStrokeCap method 199, 214
 setStrokeWidth method 199
 setStyle method 199
 styles 199
 Palette in the layout editor 48
 parent of an Activity 163
 parse method of class Uri 311
 Path class 175
 moveTo method 203
 quadTo method 204
 reset method 203
 payment processor 391
 permission in Android 6.0 (Marshmallow) 176, 181
 WRITE_EXTERNAL_PERMISSION 176
 photo sharing 400
 Photo Sphere 10
 pixel density 50

play method of class `SoundPool` 241
 Play Store app 397
 pointer (finger) in touch events 175, 198
 pointer (for touch events) 202
 pop the back stack 356
`popBackStack` method of class `FragmentManager` 356
 portrait orientation 36, 81, 102
 positive action in a dialog 118
`postDelayed` method of class `Handler` 117, 160
 PR Leap 402
 Preference class 115
 PreferenceFragment class 113, 115, 161
 `addPreferencesFromResource` method 161
 PreferenceManager class 115, 141, 142
 `setDefaultValues` method 141, 142
Preparing for Release 385
 pressure sensor 174
 prevent the soft keyboard from being displayed at app startup 298
 Preview All Screen Sizes 64, 138
 price 391
 pricing your app 390
 primary key 344
`printBitmap` method of class `PrintHelper` 206
`PrintHelper` class 206
 `printBitmap` method 206
 `SCALE_MODE_FILL` 206
 `SCALE_MODE_FIT` 206
 private key 388
 PRLog 402
 product icon
 size 387
 programmatically create GUI components 114
 progress property of a `SeekBar` 87
 ProGuard 388
 project 39
 dependencies 298

project templates
 Blank Activity 43
 Fullscreen Activity 43
 Master-Detail Application 43
 Project Volta 12
 Project window 45
 Properties window 45, 52, 55, 56, 86
 property animation 116, 129
 PROTECTION_NORMAL permissions 266
 <provider> element in `AndroidManifest.xml` 332
 proximity sensor 174
 public relations 401
 publish a new version of an app 398
 publishing data on an Android device 15
 push onto the back stack 357
`putExtra` method of class `Intent` 314
`putParcelable` method of class `Bundle` 357
`putString` method of class `SharedPreferences.Editor` 310
`putStringSet` method of class `SharedPreferences.Editor` 146

Q

quadratic bezier curve 204
`quadTo` method of class `Path` 204
 query method of a `ContentProvider` 348
 query method of class `SQLiteQueryBuilder` 350

R

R class 96
 R.drawable class 96
 R.id class 96
 R.layout class 96
 R.layout.activity_main constant 96
 R.string class 96
 raw folder of an Android project 46, 115, 116

Rect class
 offset method 236
`recycle` method of class `Bitmap` 200
 RecyclerView class 296, 307
 `addItemDecoration` method 307
 format of a list item 302
 `setAdapter` method 307
 `setLayoutManager` method 307
 RecyclerView.Adapter class 297, 307, 332
 for a `Cursor` 365
 `getItemCount` method 318, 368
 `onBindViewHolder` method 318, 368
 `onCreateViewHolder` method 318, 368
 RecyclerView.ItemDecoration class 297, 307, 332
 RecyclerView.LayoutManager class 297
 RecyclerView.ViewHolder class 297
 red method of class `Color` 209
 redraw a `View` 201
`registerListener` method of class `SensorManager` 190
`registerOnSharedPreferenceChangeListener` method of class `SharedPreferences` 142
 RelativeLayout 48
 release method of class `SoundPool` 250
 remove apps from Market 398
 rendering and tracking text 16
`replace` method of class `FragmentTransaction` 357
 reporting bugs 3
 Representational State Transfer (REST) 260
`requestFocus` method of class `View` 308
`requestPermissions` method of class `Fragment` 195, 196
 requirements 20

res folder of an Android project 46, 54
 res/drawable-mdpi folder 335
 res/raw folder of an Android project 220, 223
 reset method of class Path 203
 resolveActivity method of class Intent 119
 resource 395
 resource files 38
 resource folders
 qualified names 116
 resources
 alternative-resource naming conventions 68
 default 68
 Localization Checklist 71
 localized 68
 style 88, 331
 Resources class 143
 getConfiguration method 143, 144
 Resources dialog 55, 56, 86
 REST (Representational State Transfer) web service 260
 invoke 282
 restrict maximum number of digits in an EditText 78
 reusable software components 18
 Reuse 19
 reuse 19
 reverse engineering 388
 RGB 169
 RGB value 57
 rotate animation for a View 128
 rotation vector sensor 174
 rowCount property of a GridLayout 83
 Runnable interface 117, 247
 runOnUiThread method of class Activity 247

S
 saved state 96
 scalable vector graphic 127
 scale animation for a View 128
 scale mode 206
 SCALE_MODE_FILL 206

SCALE_MODE_FIT 206
 scale-independent pixels (sp) 56
 scaleType property of an ImageView 135
 screen capture 389
 screen capturing and sharing 12
Screen Record tool in Android Studio 390
 screen resolution 50
 screen size 50
 screen type for a GUI's design 47
 screenshot specifications 389
 ScrollView class 338
 SDK versions and API levels 42
 search operators (Twitter) 287
 SeekBar
 max property 88
 progress property 87
 SeekBar class 74, 78, 94
 SeekBar.OnSeekBarChangeListener interface 79, 94, 210
 onProgressChanged method 79
 send a message to an object 19
 Sensor class 174
 SENSOR_DELAY_NORMAL constant of class SensorManager 190
 Sensor.TYPE_ACCELEROMETER constant 190
 SensorEvent class 193
 SensorEventListener interface 192
 SensorEventListener listener 192
 SensorManager class 190
 getDefaultSensor method 190
 registerListener method 190
 unregisterListener method 191
 SensorManager.SENSOR_DELAY_NORMAL constant 190
 sensors
 accelerometer 174, 192
 gravity 174
 gyroscope 174
 light 174

sensors (cont.)
 linear acceleration 174
 magnetic field 174
 orientation 174
 pressure 174
 proximity 174
 rotation vector 174
 temperature 174
 service 76
 set in an animation 128
 Set interface 120
 setAdapter method of class ListView 279
 setAdapter method of class RecyclerView 307
 setAntiAlias method of class Paint 199
 setArguments method of class Fragment 357
 setAudioAttributes method of class
 AudioAttributes.Builder 240
 setBackground-color method 210
 setBackground-color method of class View 210
 setContentView method of class Activity 96
 setDefaultValues method of class PreferenceManager 141, 142
 setDuration method of class Animator 118
 setDuration method of class ViewAnimationUtils 157
 setHasOptionsMenu method of class Fragment 189
 setImageBitmap method of class View 214
 setImageDrawable method of class InputStream 155
 setItems method of class AlertDialog.Builder 297, 313
 setLayoutManager method of class RecyclerView 307
 setNotificationUri method of class Cursor 350

- setRepeatCount method of class Animation 117, 151
- setRequestedOrientation method of class Activity 143
- setStrokeCap method of class Paint 199, 214
- setStrokeWidth method of class Paint 199
- setStyle method of class Paint 199
- setSystemUiVisibility method of class View 253
- setTables method of a SQLiteQueryBuilder 348, 349
- setTag method of class View 264
- Setting hardware emulation options 30
- settings icon 107, 108
- setUsage method of class AudioAttributes 221, 240
- setVolumeControlStream method of class Activity 221, 226
- shades of a color 80
- shape element 335
- SharedPreferences class 115, 295, 305
 - edit method 146, 309
 - getAll method 307
 - getString method 311
 - getStringSet method 146
 - registerOnSharedPreferenceChangeListener method 142
- SharedPreferences.Editor class 115, 146, 296, 309, 310
 - apply method 146, 310
 - commit method 147
 - putString method 310
 - putStringSet method 146
- shouldShowRequestPermissionRationale method of class Fragment 195
- show method of class DialogFragment 159
- show method of class FloatingActionButton 308
- showAsAction attribute of a menu item 128, 179
- showDividers property of a LinearLayout 336
- shuffle a collection 155
- shuffle method of class Collections 120
- signing apps 386
- simple collision detection 235
- simple touch events 220
- simple_list_item_1 368
- single-screen app 43
- slider (SeekBar) 76
- smoothScrollToPosition method of class ListView 283
- SMS 293
- Snackbar class 265, 267, 280
- Social API 9
- social media sites 399
- social networking 399, 400
- soft buttons 27
- soft keyboard
 - prevent display at app startup 298
 - remain on screen 81
 - types 339
- sort
 - case insensitive 307
- sort method of class Collections 307
- sound effects 220
- sound files 223
- SoundPool class 220, 240
 - load method 241
 - play method 241
 - release method 250
- SoundPool.Builder class 221, 240
- sounds 220
- source code 2
- source-code listing 2
- sp (scale-independent pixels) 56
- SQL (Structured Query Language) 329
- SQLite 15, 324, 329
- SQLiteDatabase class 329
 - delete method 354
 - execSQL method 345
 - insert method 351
 - update method 352
- SQLiteOpenHelper class 329, 344
 - getReadableDatabase method 350
 - getWritableDatabase method 351
 - onCreate method 345
 - onDowngrade method 345
 - onUpgrade method 345
- SQLiteQueryBuilder class 348
 - appendWhere method 349
 - join 349
 - query method 350
 - setTables method 348, 349
- src property of a ImageView 53, 62
- star ratings for apps 398
- start method of class Animator 118
- startActivity method of class Context 311
- startAnimation method of class View 117
- stateAlwaysHidden (virtual keyboard mode) 298
- states of an Activity 77
- stream for playing music 226
- @string resource 55
- String resource
 - containing multiple format specifiers 124
- string resource 55
- String.CASE_INSENSITIVE_OR_DER.Comparator<String> object 307
- strings.xml 55
- stroke element of a shape 335
- Structured Query Language (SQL) 329
- style (define new) 334
- style attribute of a GUI component 331
- style property of a View 339
- style property of a view 91
- style resource 331, 339
- style resources 88
- styles.xml 334
- support library
 - FragmentManager 114

surfaceChanged method of interface SurfaceHolder. Callback **250**

surfaceCreated method of interface SurfaceHolder. Callback **250**

surfaceDestroyed method of interface SurfaceHolder.Callback **250**

SurfaceHolder class **221, 240**
 addCallback method **240**
 lockCanvas method **253**

SurfaceHolder.Callback interface **221, 240, 250**
 surfaceChanged method **250**
 surfaceCreated method **250**
 surfaceDestroyed method **250**

SurfaceView class **221, 240**
 getHolder method **240**

synchronized **253**

syntax coloring **xv, 2**

system bar **37**

SYSTEM_UI_FLAG_IMMERSIVE **254**

T

tablet **8**

TalkBack **36, 38, 66, 135**
 enable/disable **66**
 Localization **71**

temperature sensor **174**

text box **78**

text field **78**

text property of a TextView **53, 55, 134**

Text tab in the layout editor **37**

textAppearance property of a TextView **84**

textColor property of a Button **137**

textColor property of a TextView **58**

textSize property of a TextView **56, 136**

textStyle property of a TextView **136**

Text-to-Speech API **9**

TextView
 gravity property **58, 88**
 text property **53, 55**
 textAppearance property **84**
 textColor property **58**
 textSize property **56**

TextView class **38, 54, 78, 94**
 hint property **86**
 text property **134**
 textSize property **136**
 textStyle property **136**

TextView component **49**

TextChanged interface **79, 93, 307**
 onTextChanged method **79**

theme **88**

Theme Editor **80**

Theme.AppCompat.Light.DarkActionBar **79, 88**

thread (for animation) **221**

Thread class **252**

Threadr class **221**

Tip Calculator app **17**

Toast class **117, 147**
 makeText method **147**

Tools: logcat **119**

tooltip in layout editor **61**

touch event **175, 202**

touch events
 simple **220**

track app installs **398**

training from Deitel **404**

translate animation
 android:duration attribute **129**
 android:fromXDelta attribute **129**
 android:startOffset attribute **129**
 android:toXDelta attribute **129**

translate animation for a View **128**

tweened animation **116, 128**

tweet **400**

Twitter **5, 293, 400**
 @deitel **400**
 hashtag **400**
 tweet **400**

Twitter Fabric (mobile development platform) **320**

Twitter search **287**
 operators **288**

TYPE_ACCELEROMETER constant of class Sensor **190**

U

UI thread **117**

Uniform Resource Identifier (URI) **311**

Uniform Resource Locator (URL) **311**

unique identifier for an app **40**

unregisterListener method of class SensorManager **191**

up button **131, 161**

update method of a ContentProvider **352**

update method of a ContentResolver **374**

update method of class SQLiteDatabase **352**

URI (Uniform Resource Identifier) **311**

Uri class **311, 330**
 getLastPathSegment method **349**
 parse method **311**

UriMatcher class **346**

URL (Uniform Resource Locator) **311**

URL encoded String **311**

USB debugging **30**

useDefaultMargins property of a GridLayout **83**

utilities **31**

V

values folder of an Android project **47, 55**

vector asset
 add to project **127**

Vector Asset Studio **113, 127, 178**

version code **387**

version name **387**

VERSION_SDK_INT **253**

versioning your app **386**
Versioning Your Applications **387**

video 15
 video sharing 400
 view 76
 GUI component 37
 view (GUI component) 46
 View animations 128
 View class 76, 210, 221
 contentDescription
 property 135
 custom subclass 237
 getTag method 264
 invalidate method 200
 layout:gravity property 134
 layout:margin property 134
 layout:weight property 134
 onDraw method 201
 onSizeChanged method 200, 241
 onTouchEvent method 175, 202, 220, 251
 redraw a View 201
 requestFocus method 308
 setImageBitmap method 214
 setSystemUiVisibility method 253
 setTag method 264
 size changes 241
 startAnimation method 117
 View.OnClickListener interface 151
 View.OnLongClickListener interface 311
 View.SYSTEM_UI_FLAG_IMMERSIVE 254
 ViewAnimationUtils class 118
 createCircularReveal method 118, 157
 setDuration method 157

ViewGroup class 338
 view-holder pattern 264
 view-holder pattern 297
 views
 ImageView 38, 49, 60
 TextView 38, 49, 54
 viral marketing 399, 400
 viral video 400
 virtual camera operator 9
 virtual goods 393
 virtual keyboard mode
 stateAlwaysHidden 298
 visual impairment 36
 Voice Interaction API 13
 volume 221

W

WeatherBug 6
 web service 259
 API key 260
 host 259
 REST 260
 web services 5
 Amazon eCommerce 6
 eBay 6
 Facebook 6
 Flickr 6
 Foursquare 6
 Google Maps 6
 Instagram 6
 LinkedIn 6
 Microsoft Bing 6
 Netflix 6
 PayPal 6
 Salesforce.com 6
 Skype 6
 Twitter 6
 Wikipedia 6
 Yahoo Search 6
 YouTube 6
 Zillow 6

weightSum property of a
 LinearLayout 336
 Welcome app 17
 Welcome window in Android
 Studio 39
 widget 16, 94
 width of a column 81
 Wi-Fi Direct 9
 wildcard in a Uri 347
 Windows 16
 windowSoftInputMode option 298
 WRITE_EXTERNAL_PERMISSION 176
 www.deitel.com/training 404

X

XML 49
 xml folder of an Android project 46, 115, 116
 XML namespace
 android 179
 app 179
 XML resource files 38
 XML utilities 16

Y

YouTube 390