



DEITEL DEVELOPER SERIES

SECOND EDITION

Android™ for Programmers

An App-Driven Approach

7 Fully Coded
Android™ Apps

Volume 1



PAUL DEITEL • HARVEY DEITEL
ABBEY DEITEL

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



ANDROID[™] FOR PROGRAMMERS
AN APP-DRIVEN APPROACH
SECOND EDITION, VOLUME 1
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

© 2014 Pearson Education, Inc.

Portions of the cover are modifications based on work created and shared by Google (<http://code.google.com/policies.html>) and used according to terms described in the Creative Commons 3.0 Attribution License (<http://creativecommons.org/licenses/by/3.0/>).

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

ISBN-13: 978-0-13357092-2

ISBN-10: 0-13-357092-4

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

First printing, December 2013

ANDROID™ FOR PROGRAMMERS

AN APP-DRIVEN APPROACH

SECOND EDITION, VOLUME 1

DEITEL® DEVELOPER SERIES

Paul Deitel • Harvey Deitel • Abbey Deitel
Deitel & Associates, Inc.



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

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 screen shots 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. Screen shots 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.

*In Memory of Amar G. Bose, MIT Professor and
Founder and Chairman of the Bose Corporation:*

*It was a privilege being your student—and members
of the next generation of Deitels, who heard our dad
say how your classes inspired him to do his best work.
You taught us that if we go after the really hard prob-
lems, then great things can happen.*

*Harvey Deitel
Paul and Abbey Deitel*

This page intentionally left blank



Contents

Preface

xiv

Before You Begin

xxiii

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	7
1.4.1	Android 2.2 (Froyo)	7
1.4.2	Android 2.3 (Gingerbread)	8
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
1.5	Downloading Apps from Google Play	11
1.6	Packages	12
1.7	Android Software Development Kit (SDK)	13
1.8	Object-Oriented Programming: A Quick Refresher	16
1.8.1	The Automobile as an Object	17
1.8.2	Methods and Classes	17
1.8.3	Instantiation	17
1.8.4	Reuse	17
1.8.5	Messages and Method Calls	17
1.8.6	Attributes and Instance Variables	18
1.8.7	Encapsulation	18
1.8.8	Inheritance	18
1.8.9	Object-Oriented Analysis and Design (OOAD)	18
1.9	Test-Driving the Doodlz App in an Android Virtual Device (AVD)	19
1.9.1	Running the Doodlz App in the Nexus 4 Smartphone AVD	19
1.9.2	Running the Doodlz App in a Tablet AVD	28
1.9.3	Running the Doodlz 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® the Android Developer Tools: Introducing Visual GUI Design, Layouts, Accessibility and Internationalization

2.1	Introduction	36
2.2	Technologies Overview	37
2.2.1	Android Developer Tools IDE	37
2.2.2	TextViews and ImageViews	37
2.2.3	App Resources	37
2.2.4	Accessibility	37
2.2.5	Internationalization	37
2.3	Creating an App	38
2.3.1	Launching the Android Developer Tools IDE	38
2.3.2	Creating a New Project	38
2.3.3	New Android Application Dialog	39
2.3.4	Configure Project Step	40
2.3.5	Configure Launcher Icon Step	40
2.3.6	Create Activity Step	42
2.3.7	Blank Activity Step	43
2.4	Android Developer Tools Window	44
2.4.1	Package Explorer Window	45
2.4.2	Editor Windows	45
2.4.3	Outline Window	45
2.4.4	App Resource Files	45
2.4.5	Graphical Layout Editor	46
2.4.6	The Default GUI	46
2.5	Building the App's GUI with the Graphical Layout Editor	48
2.5.1	Adding Images to the Project	48
2.5.2	Changing the Id Property of the RelativeLayout and the TextView	49
2.5.3	Configuring the TextView	50
2.5.4	Adding ImageViews to Display the Images	54
2.6	Running the Welcome App	56
2.7	Making Your App Accessible	57
2.8	Internationalizing Your App	59
2.9	Wrap-Up	63

3 Tip Calculator App 64

Introducing GridLayout, LinearLayout, EditText, SeekBar, Event Handling, NumberFormat and Defining App Functionality with Java

3.1	Introduction	65
3.2	Test-Driving the Tip Calculator App	66
3.3	Technologies Overview	67
3.3.1	Class Activity	67
3.3.2	Activity Lifecycle Methods	67
3.3.3	Arranging Views with LinearLayout and GridLayout	68

3.3.4	Creating and Customizing the GUI with the Graphical Layout Editor and the Outline and Properties Windows	68
3.3.5	Formatting Numbers as Locale-Specific Currency and Percentage Strings	69
3.3.6	Implementing Interface <code>TextWatcher</code> for Handling <code>EditText</code> Text Changes	69
3.3.7	Implementing Interface <code>OnSeekBarChangeListener</code> for Handling <code>SeekBar</code> Thumb Position Changes	69
3.3.8	<code>AndroidManifest.xml</code>	70
3.4	Building the App's GUI	70
3.4.1	<code>GridLayout</code> Introduction	70
3.4.2	Creating the <code>TipCalculator</code> Project	72
3.4.3	Changing to a <code>GridLayout</code>	72
3.4.4	Adding the <code>TextViews</code> , <code>EditText</code> , <code>SeekBar</code> and <code>LinearLayouts</code>	73
3.4.5	Customizing the Views to Complete the Design	75
3.5	Adding Functionality to the App	79
3.6	<code>AndroidManifest.xml</code>	87
3.7	Wrap-Up	88

4 Twitter® Searches App 89

SharedPreferences, Collections, ImageButton, ListView, ListActivity, ArrayAdapter, Implicit Intents and AlertDialogs

4.1	Introduction	90
4.2	Test-Driving the App	91
4.2.1	Importing the App and Running It	91
4.2.2	Adding a Favorite Search	92
4.2.3	Viewing Twitter Search Results	93
4.2.4	Editing a Search	94
4.2.5	Sharing a Search	96
4.2.6	Deleting a Search	96
4.2.7	Scrolling Through Saved Searches	97
4.3	Technologies Overview	97
4.3.1	<code>ListView</code>	97
4.3.2	<code>ListActivity</code>	98
4.3.3	Customizing a <code>ListActivity</code> 's Layout	98
4.3.4	<code>ImageButton</code>	98
4.3.5	<code>SharedPreferences</code>	98
4.3.6	Intents for Launching Other Activities	99
4.3.7	<code>AlertDialog</code>	99
4.3.8	<code>AndroidManifest.xml</code>	100
4.4	Building the App's GUI	100
4.4.1	Creating the Project	100
4.4.2	<code>activity_main.xml</code> Overview	101
4.4.3	Adding the <code>GridLayout</code> and Components	102
4.4.4	Graphical Layout Editor Toolbar	107

x Contents

4.4.5	ListView Item's Layout: list_item.xml	108
4.5	Building the MainActivity Class	109
4.5.1	package and import Statements	109
4.5.2	Extending ListActivity	111
4.5.3	Fields of Class MainActivity	111
4.5.4	Overriding Activity Method onCreate	112
4.5.5	Anonymous Inner Class That Implements the saveButton's OnClickListener to Save a New or Updated Search	114
4.5.6	addTaggedSearch Method	116
4.5.7	Anonymous Inner Class That Implements the ListView's OnItemClickListener to Display Search Results	117
4.5.8	Anonymous Inner Class That Implements the ListView's OnItemLongClickListener to Share, Edit or Delete a Search	119
4.5.9	shareSearch Method	121
4.5.10	deleteSearch Method	122
4.6	AndroidManifest.xml	124
4.7	Wrap-Up	124

5 Flag Quiz App

125

Fragments, Menus, Preferences, AssetManager, Tweened Animations, Handler, Toasts, Explicit Intents, Layouts for Multiple Device Orientations

5.1	Introduction	126
5.2	Test-Driving the Flag Quiz App	128
5.2.1	Importing the App and Running It	128
5.2.2	Configuring the Quiz	128
5.2.3	Taking the Quiz	130
5.3	Technologies Overview	132
5.3.1	Menus	132
5.3.2	Fragments	132
5.3.3	Fragment Lifecycle Methods	133
5.3.4	Managing Fragments	133
5.3.5	Preferences	133
5.3.6	assets Folder	133
5.3.7	Resource Folders	134
5.3.8	Supporting Different Screen Sizes and Resolutions	134
5.3.9	Determining the Screen Size	135
5.3.10	Toasts for Displaying Messages	135
5.3.11	Using a Handler to Execute a Runnable in the Future	135
5.3.12	Applying an Animation to a View	135
5.3.13	Logging Exception Messages	136
5.3.14	Using an Explicit Intent to Launch Another Activity in the Same App	136
5.3.15	Java Data Structures	136
5.4	Building the GUI and Resource Files	136
5.4.1	Creating the Project	136

5.4.2	strings.xml and Formatted String Resources	137
5.4.3	arrays.xml	138
5.4.4	colors.xml	139
5.4.5	dimens.xml	139
5.4.6	activity_settings.xml Layout	140
5.4.7	activity_main.xml Layout for Phone and Tablet Portrait Orientation	140
5.4.8	fragment_quiz.xml Layout	140
5.4.9	activity_main.xml Layout for Tablet Landscape Orientation	143
5.4.10	preferences.xml for Specifying the App's Settings	144
5.4.11	Creating the Flag Shake Animation	145
5.5	MainActivity Class	147
5.5.1	package Statement, import Statements and Fields	147
5.5.2	Overridden Activity Method onCreate	148
5.5.3	Overridden Activity Method onStart	150
5.5.4	Overridden Activity Method onCreateOptionsMenu	150
5.5.5	Overridden Activity Method onOptionsItemSelected	151
5.5.6	Anonymous Inner Class That Implements OnSharedPreferencesChangeListener	152
5.6	QuizFragment Class	153
5.6.1	package Statement and import Statements	153
5.6.2	Fields	154
5.6.3	Overridden Fragment Method onCreateView	155
5.6.4	Method updateGuessRows	157
5.6.5	Method updateRegions	158
5.6.6	Method resetQuiz	158
5.6.7	Method loadNextFlag	160
5.6.8	Method getCountryName	162
5.6.9	Anonymous Inner Class That Implements OnClickListener	162
5.6.10	Method disableButtons	165
5.7	SettingsFragment Class	165
5.8	SettingsActivity Class	166
5.9	AndroidManifest.xml	166
5.10	Wrap-Up	167

6 Cannon Game App **168**

Listening for Touches, Manual Frame-By-Frame Animation, Graphics, Sound, Threading, SurfaceView and SurfaceHolder

6.1	Introduction	169
6.2	Test-Driving the Cannon Game App	171
6.3	Technologies Overview	171
6.3.1	Attaching a Custom View to a Layout	171
6.3.2	Using the Resource Folder raw	171
6.3.3	Activity and Fragment Lifecycle Methods	171
6.3.4	Overriding View Method onTouchEvent	172

6.3.5	Adding Sound with SoundPool and AudioManager	172
6.3.6	Frame-by-Frame Animation with Threads, SurfaceView and SurfaceHolder	172
6.3.7	Simple Collision Detection	173
6.3.8	Drawing Graphics Using Paint and Canvas	173
6.4	Building the App's GUI and Resource Files	173
6.4.1	Creating the Project	173
6.4.2	strings.xml	174
6.4.3	fragment_game.xml	174
6.4.4	activity_main.xml	175
6.4.5	Adding the Sounds to the App	175
6.5	Class Line Maintains a Line's Endpoints	175
6.6	MainActivity Subclass of Activity	176
6.7	CannonGameFragment Subclass of Fragment	176
6.8	CannonView Subclass of View	178
6.8.1	package and import Statements	178
6.8.2	Instance Variables and Constants	179
6.8.3	Constructor	180
6.8.4	Overriding View Method onSizeChanged	182
6.8.5	Method newGame	183
6.8.6	Method updatePositions	184
6.8.7	Method fireCannonball	187
6.8.8	Method alignCannon	188
6.8.9	Method drawGameElements	189
6.8.10	Method showGameOverDialog	191
6.8.11	Methods stopGame and releaseResources	192
6.8.12	Implementing the SurfaceHolder.Callback Methods	193
6.8.13	Overriding View Method onTouchEvent	194
6.8.14	CannonThread: Using a Thread to Create a Game Loop	195
6.9	Wrap-Up	196

7 **Doodlz App** **198**

Two-Dimensional Graphics, Canvas, Bitmap, Accelerometer, SensorManager, Multitouch Events, MediaStore, Printing, Immersive Mode

7.1	Introduction	199
7.2	Technologies Overview	201
7.2.1	Using SensorManager to Listen for Accelerometer Events	201
7.2.2	Custom DialogFragments	201
7.2.3	Drawing with Canvas and Bitmap	202
7.2.4	Processing Multiple Touch Events and Storing Lines in Paths	202
7.2.5	Android 4.4 Immersive Mode	202
7.2.6	GestureDetector and SimpleOnGestureListener	202
7.2.7	Saving the Drawing to the Device's Gallery	202
7.2.8	Android 4.4 Printing and the Android Support Library's PrintHelper Class	203

7.3	Building the App’s GUI and Resource Files	203
7.3.1	Creating the Project	203
7.3.2	strings.xml	203
7.3.3	dimens.xml	204
7.3.4	Menu for the DoodleFragment	205
7.3.5	activity_main.xml Layout for MainActivity	206
7.3.6	fragment_doodle.xml Layout for DoodleFragment	206
7.3.7	fragment_color.xml Layout for ColorDialogFragment	207
7.3.8	fragment_line_width.xml Layout for LineWidthDialogFragment	209
7.3.9	Adding Class EraseImageDialogFragment	210
7.4	MainActivity Class	211
7.5	DoodleFragment Class	212
7.6	DoodleView Class	219
7.7	ColorDialogFragment Class	231
7.8	LineWidthDialogFragment Class	234
7.9	EraseImageDialogFragment Class	238
7.10	Wrap-Up	239

8 Address Book App **241**

ListFragment, FragmentTransactions and the Fragment Back Stack, Threading and AsyncTasks, CursorAdapter, SQLite and GUI Styles

8.1	Introduction	242
8.2	Test-Driving the Address Book App	245
8.3	Technologies Overview	245
8.3.1	Displaying Fragments with FragmentTransactions	246
8.3.2	Communicating Data Between a Fragment and a Host Activity	246
8.3.3	Method onSaveInstanceState	246
8.3.4	Defining Styles and Applying Them to GUI Components	246
8.3.5	Specifying a Background for a TextView	246
8.3.6	Extending Class ListFragment to Create a Fragment That Contains a ListView	247
8.3.7	Manipulating a SQLite Database	247
8.3.8	Performing Database Operations Outside the GUI Thread with AsyncTasks	247
8.4	Building the GUI and Resource Files	247
8.4.1	Creating the Project	247
8.4.2	Creating the App’s Classes	248
8.4.3	strings.xml	248
8.4.4	styles.xml	249
8.4.5	textview_border.xml	250
8.4.6	MainActivity’s Layout: activity_main.xml	251
8.4.7	DetailsFragment’s Layout: fragment_details.xml	251
8.4.8	AddEditFragment’s Layout: fragment_add_edit.xml	253
8.4.9	Defining the Fragments’ Menus	254
8.5	MainActivity Class	255

8.6	ContactListFragment Class	261
8.7	AddEditFragment Class	268
8.8	DetailsFragment Class	274
8.9	DatabaseConnector Utility Class	282
8.10	Wrap-Up	287

9 **Google Play and App Business Issues** **289**

9.1	Introduction	290
9.2	Preparing Your Apps for Publication	290
9.2.1	Testing Your App	291
9.2.2	End User License Agreement	291
9.2.3	Icons and Labels	291
9.2.4	Versioning Your App	292
9.2.5	Licensing to Control Access to Paid Apps	292
9.2.6	Obfuscating Your Code	292
9.2.7	Getting a Private Key for Digitally Signing Your App	293
9.2.8	Screenshots	293
9.2.9	Promotional App Video	294
9.3	Pricing Your App: Free or Fee	295
9.3.1	Paid Apps	296
9.3.2	Free Apps	296
9.4	Monetizing Apps with In-App Advertising	297
9.5	Monetizing Apps: Using In-App Billing to Sell Virtual Goods	298
9.6	Registering at Google Play	299
9.7	Setting Up a Google Wallet Merchant Account	300
9.8	Uploading Your Apps to Google Play	301
9.9	Launching the Play Store from Within Your App	302
9.10	Managing Your Apps in Google Play	303
9.11	Other Android App Marketplaces	303
9.12	Other Popular Mobile App Platforms	303
9.13	Marketing Your Apps	304
9.14	Wrap-Up	308

Index **310**



Preface

Welcome to the dynamic world of Android smartphone and tablet app development with the Android Software Development Kit (SDK), the Java™ programming language, the Eclipse-based Android Development Tools IDE, and the new and rapidly evolving Android Studio IDE.

Android for Programmers: An App-Driven Approach, 2/e, Volume 1 presents leading-edge mobile computing technologies for professional software developers. At the heart of the book is our *app-driven approach*—we present concepts in the context of *seven complete working Android apps* rather than using code snippets. Chapters 2–8 each present one app. We begin each of these chapters with an introduction to the app, an app test-drive showing one or more sample executions and a technologies overview. Then we proceed with a detailed code walkthrough of the app’s source code. All of the source code is available at www.deitel.com/books/AndroidFP2. We recommend that you have the source code open in the IDE as you read the book.

Sales of Android devices and app downloads have been growing exponentially. The first-generation Android phones were released in October 2008. A study by Strategy Analytics showed that by October 2013, Android had 81.3% of the global smartphone market share, compared to 13.4% for Apple, 4.1% for Microsoft and 1% for BlackBerry.¹ According to an IDC report, by the end of the first quarter of 2013 Android had 56.5% of the global tablet market share, compared to 39.6% for Apple’s iPad and 3.7% for Microsoft Windows tablets.²

There are now over one billion Android smartphones and tablets in use,³ and more than 1.5 million Android devices are being activated daily.⁴ According to IDC, Samsung is the leading Android manufacturer, accounting for nearly 40% of Android device shipments in the third quarter of 2013.

Billions of apps have been downloaded from Google Play™—Google’s marketplace for Android apps. The opportunities for Android app developers are enormous.

Fierce competition among popular mobile platforms and carriers is leading to rapid innovation and falling prices. Competition among the dozens 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

1. <http://blogs.strategyanalytics.com/WSS/post/2013/10/31/Android-Captures-Record-81-Percent-Share-of-Global-Smartphone-Shipments-in-Q3-2013.aspx>.
2. <http://www.idc.com/getdoc.jsp?containerId=prUS24093213>.
3. <http://www.android.com/kitkat>.
4. <http://www.technobuffalo.com/2013/04/16/google-daily-android-activations-1-5-million>.

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, 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. 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. Because of the improved Android development tools, we were able to eliminate almost all XML markup in this edition. There are still two small, easy-to-understand XML files you'll need to manipulate. We use only complete, working apps, so if you don't know Java but have object-oriented programming experience in languages like C#/.NET, Objective-C/Cocoa or C++ (with class libraries), 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, but it presents a significant amount of Java in the context of Android app development. If you're interested in learning Java, check out our publications:

- *Java for Programmers, 2/e* (www.deitel.com/books/javafp2)
- *Java Fundamentals: Parts I and II* LiveLessons videos (www.deitel.com/books/LiveLessons).
- *Java How to Program, 10/e* (www.deitel.com/books/jhtp10)

If you're not familiar with XML, see these online tutorials:

- <http://www.ibm.com/developerworks/xml/newto>
- http://www.w3schools.com/xml/xml_what_is.asp
- http://www.deitel.com/articles/xml_tutorials/20060401/XMLBasics
- http://www.deitel.com/articles/xml_tutorials/20060401/XMLStructuringData

Key Features

Here are some of this book's key features:

App-Driven Approach. Chapters 2–8 each present one completely coded app—we discuss what the app does, show screen shots of the app in action, test-drive it and overview the technologies and architecture we'll use 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 SDK 4.3 and 4.4. We cover various new Android Software Development Kit (SDK) 4.3 and 4.4 features.

Fragments. Starting with Chapter 5, we use Fragments to create and manage portions of each app's GUI. You can combine several fragments to create user interfaces that take ad-

vantage of tablet screen sizes. You also can easily interchange fragments to make your GUIs more dynamic, as you'll do in Chapter 8.

Support for multiple screen sizes and resolutions. Throughout the app chapters we demonstrate how to use Android's mechanisms for automatically choosing resources (layouts, images, etc.) based on a device's size and orientation.

Eclipse-Based Android Development Tools (ADT) IDE coverage in the print book. The free Android Development Tools (ADT) integrated development environment (IDE)—which includes Eclipse and the ADT plugin—combined with 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.

Android Studio IDE. This is the preferred IDE for the future of Android app development. Because it's new and evolving rapidly, we put our discussions of it online at:

<http://www.deitel.com/books/AndroidFP2>

We'll show how to import existing projects so you can test-drive our apps. We'll also demonstrate how to create new apps, build GUIs, modify resource files and test your apps. If you have any questions, contact us at deitel@deitel.com.

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.

Printing Framework. Android 4.4 KitKat allows you to add printing functionality to your apps, such as locating available printers over Wi-Fi or the cloud, selecting the paper size and specifying which pages to print.

Testing on Android Smartphones, Tablets and the Android Emulator. For the best app-development experience, 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.

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

Uploading Apps to Google Play. Chapter 9, Google Play and App Business Issues, walks you through the registration process for 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 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 9 can be read after Chapter 1.

Features

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

comments appear in green
 keywords appear in dark blue
 constants and literal values appear in light blue
 all other code appears in non-bold black

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 in **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:

www.deitel.com/books/AndroidFP2
www.informit.com/title/0133570924

Documentation. All the Android and Java documentation you'll need to develop Android apps is available free at <http://developer.android.com> and <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. The documentation for Eclipse is available at www.eclipse.org/documentation. The documentation for Android Studio is available at <http://developer.android.com/sdk/installing/studio.html>.

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

Figures. Hundreds of tables, source code listings and Android screen shots are included.

Software Engineering. We stress program clarity and performance, and 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 in the book is highlighted in the index in **bold maroon**.

Working with Open-Source Apps

There are numerous free, open-source Android apps available online which 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 for Programmers: An App-Driven Approach, Second Edition, Volume 2

Volume 2, which will be published in 2014, contains additional app-development chapters that introduce property animation, Google Play game services, video, speech synthesis and recognition, GPS, the Maps API, the compass, object serialization, web services, audio recording and playback, Bluetooth®, HTML5 mobile apps and more. **For the status of Volume 2 and for continuing book updates, visit**

<http://www.deitel.com/books/AndroidFP2>

Android Fundamentals, Second Edition LiveLessons Video Training Products

Our *Android Fundamentals, Second Edition* LiveLessons videos show you what you need to know to start building robust, powerful Android apps with the Android Software Development Kit (SDK) 4.3 and 4.4, the Java™ programming language and the Eclipse™ and Android Studio integrated development environments (IDEs). It will include approximately 20 hours of expert training synchronized with *Android for Programmers, Second Edition* (Volumes 1 and 2). The videos for Volume 1 will be available spring 2014. For additional information about Deitel LiveLessons video products, visit

www.deitel.com/livelessons

or contact us at deitel@deitel.com. You can also access our LiveLessons videos if you have a subscription to Safari Books Online (www.safaribooksonline.com).

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 onsite training courses, partner offers and more, join the Deitel social networking communities on Facebook® (<http://www.deitel.com/deitelfan>), Twitter® (@deitel), LinkedIn® (<http://bit.ly/DeitelLinkedIn>) Google+™ (<http://google.com/+DeitelFan>), and YouTube® (<http://youtube.com/user/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 on:

www.deitel.com/books/AndroidFP2

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

Visit www.deitel.com to:

- Download code examples
- Check out the growing list of programming Resource Centers
- Receive updates for this e-book, subscribe to the free *Deitel® Buzz Online* e-mail newsletter at 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.

This book was a cooperative effort between professional and academic divisions of Pearson. We appreciate the efforts and 18-year mentorship of our friend and professional colleague Mark L. Taub, Editor-in-Chief of the Pearson Technology Group. Mark and his team handle all of our professional books and LiveLessons video products. Kim Boedigheimer recruited distinguished members of the Android community and managed the review team for the Android content. We selected the cover art and Chuti Prasertsith and Sandra Schroeder designed the cover. John Fuller manages the production of all of our Deitel Developer Series books.

We also appreciate the guidance, wisdom and energy of Tracy Johnson, Executive Editor, Computer Science. Tracy and her team handle all of our academic textbooks. Carole Snyder recruited the book's academic reviewers and managed the review process. Bob Engelhardt manages the production of our academic publications.

We'd like to thank Michael Morgano, a former colleague of ours at Deitel & Associates, Inc., now an Android developer at Imerj™, 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.

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

We wish to acknowledge the efforts of our first and second edition reviewers. 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 and Android App Developer, 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 Univer-

sity of Alcalá in Madrid), Sebastian Nykopp (Chief Architect, Reaktor), Michael Pardo (Android Developer, Mobiata), Ronan “Zero” Schwarz (CIO, OpenIntents), Arijit Sen Gupta (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 for Programmers: An App-Driven Approach, Second Edition, Volume 1* will quickly get you developing Android apps. We hope you enjoy reading the book as much as we enjoyed writing it!

Paul Deitel
Harvey Deitel
Abbey 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 certifications, and is an Oracle Java Champion. 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 Higher Education, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey M. 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 more than 50 years of experience in computing. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University. In the 1960s, through Advanced Computer Techniques and Computer Usage Corporation, he worked on the teams building various IBM operating systems. In the 1970s, he built commercial software systems. 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 Deitel. The Deitels’ publications have earned international recognition, with translations published in Simplified Chinese, Traditional Chinese, Korean, Japanese, German, Russian, Spanish, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to corporate, academic, government and military clients.

Abbey Deitel, President of Deitel & Associates, Inc., is a graduate of Carnegie Mellon University’s Tepper School of Management where she received a B.S. in Industrial Management. Abbey has been managing the business operations of Deitel & Associates, Inc. for 16 years. She has contributed to numerous Deitel & Associates publications and, together with Paul and Harvey, is the co-author of *Android for Programmers: An App-Driven Approach, 2/e*, *iPhone for Programmers: An App-Driven Approach*, *Internet & World Wide Web How to Program, 5/e*, *Visual Basic 2012 How to Program, 6/e* and *Simply Visual Basic 2010, 5/e*.

Deitel® Dive-Into® Series Corporate Training

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, Objective-C and iOS app development, Java™, C++, Visual C++®, C, Visual C#®, Visual Basic®, XML®, Python®, object technology, Internet and web programming and a growing list of additional programming and software development courses.

Through its 37-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:

www.deitel.com/training

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

Individuals wishing to purchase Deitel books and *LiveLessons* video training can do so through www.deitel.com. Bulk orders by corporations, the government, the military and academic institutions should be placed directly with Pearson. For more information, visit

www.informit.com/store/sales.aspx



Before You Begin

In this section, you'll set up your computer for use with this book. The Android development tools are frequently updated. Before reading this section, check the book's website

<http://www.deitel.com/books/AndroidFP2/>

to see if we've posted an updated version.

Font and Naming Conventions

We use fonts to distinguish between on-screen components (such as menu names and menu items) and Java code or commands. Our convention is to show on-screen components in a sans-serif bold **Helvetica** font (for example, **Project** menu) and to show file names, Java code and commands in a sans-serif **Lucida** font (for example, the keyword `public` or class `Activity`). When specifying commands to select in menus, we use the `>` notation to indicate a menu item to select. For example, **Window > Preferences** indicates that you should select the **Preferences** menu item from the **Window** menu.

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>

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 SDK/ADT Bundle based on the Eclipse IDE
- Android SDK versions 4.3 and 4.4

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

Installing the Java Development Kit (JDK)

Android requires the *Java Development Kit (JDK)* version 7 (JDK 7) or 6 (JDK 6). We used *JDK 7*. To download the JDK for Windows, OS X or Linux, go to

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

You need only the JDK. Choose the 32-bit or 64-bit version based on your computer hardware and operating system. Most recent computers have 64-bit hardware—check your system's specifications. If you have a 32-bit operating system, you must use the 32-bit JDK. Be sure to follow the installation instructions at

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

Android Integrated Development Environment (IDE) Options

Google now provides two Android IDE options:

- Android SDK/ADT bundle—a version of the *Eclipse IDE* that comes preconfigured with the latest Android Software Development Kit (SDK) and the latest Android Development Tools (ADT) plugin. At the time of this writing, these were Android SDK version 4.4 and ADT version 22.3.
- Android Studio—Google’s new Android IDE based on IntelliJ® IDEA and their preferred future IDE.

The Android SDK/ADT bundle has been widely used in Android app development for several years. Android Studio, introduced in May 2013, is an *early access version* and will be evolving rapidly. For this reason, we’ll stay with the widely used Android SDK/ADT bundle in the book, and as online supplements at

<http://www.deitel.com/books/AndroidFP2>

we’ll provide Android Studio versions of the Chapter 1 Test-Drive section and the Building the GUI section for each app, as appropriate.

Installing the Android SDK/ADT Bundle

To download the Android SDK/ADT bundle, go to

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

and click the **Download the SDK ADT Bundle** button. When the download completes, extract the ZIP file’s contents to your system. The resulting folder has an `eclipse` subfolder containing the Eclipse IDE and an `sdk` subfolder containing the Android SDK. As with the JDK, you can choose a 32-bit or 64-bit version. The Android SDK/ADT bundle 32-bit version should be used with the 32-bit JDK, and the 64-bit version with the 64-bit JDK.

Installing Android Studio



The IDE instructions in the printed book use the Android SDK/ADT bundle. You can also optionally install and use Android Studio. To download Android Studio, go to

<http://developer.android.com/sdk/installing/studio.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. [*Note:* For Android 4.4 development in Android Studio, Android now supports Java SE 7 language features, including the diamond operator, multi-catch, Strings in `switch` and `try-with-resources`.]

Set the Java Compiler Compliance Level and Show Line Numbers



Android does not fully support Java SE 7. To ensure that the book’s examples compile correctly, configure Eclipse to produce files that are compatible with Java SE 6 by performing the following steps:

1. Open Eclipse ( or ) , which is located in the `eclipse` subfolder of the Android SDK/ADT bundle’s installation folder.
2. When the **Workspace Launcher** window appears, click **OK**.

3. Select **Window > Preferences** to display the **Preferences** window. On Mac OS X, select **ADT > Preferences...**
4. Expand the **Java** node and select the **Compiler** node. Under **JDK Compliance**, set the **Compiler compliance level** to 1.6 (to indicate that Eclipse should produce compiled code that's compatible with Java SE 6).
5. Expand the **General > Editors** node and select **TextEditors**, then ensure that **Show line numbers** is selected and click **OK**.
6. Close Eclipse.

Android 4.3 SDK

This book's examples were written using the Android 4.3 and 4.4 SDKs. At the time of this writing, 4.4 was the version included with the Android SDK/ADT bundle and Android Studio. You should also install Android 4.3 (and any other versions you might want to support in your apps). To install other Android platform versions, perform the following steps (skipping Steps 1 and 2 if Eclipse is already open):

1. Open Eclipse. Depending on your platform, the icon will appear as  or .
2. When the **Workspace Launcher** window appears, click **OK**.
3. On Mac OS X, if you see a window indicating "Could not find SDK folder '/Users/YourAccount/android-sdk-macosx'," click **Open Preferences** then **Browse...** and select the **sdk** folder located where you extracted the Android SDK/ADT bundle.
4. Select **Window > Android SDK Manager** to display the **Android SDK Manager** (Fig. 1).

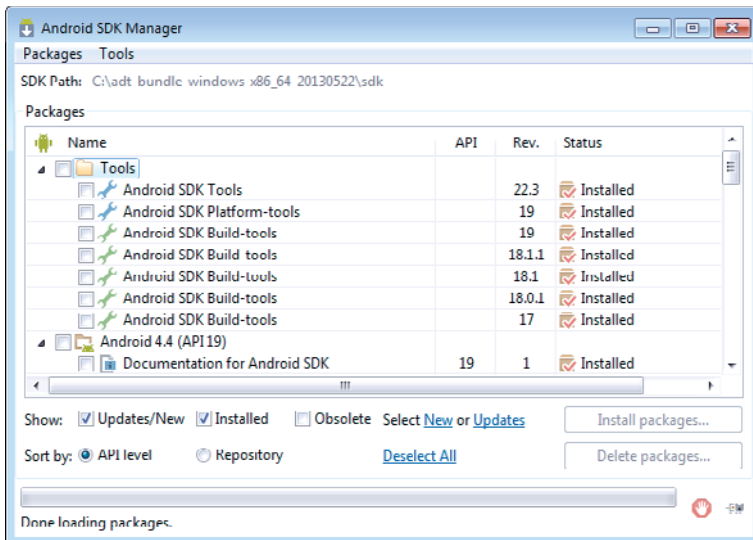


Fig. 1 | Android SDK Manager window.

5. The **Android SDK Manager's** **Name** column shows all of the tools, platform versions and extras (such as APIs for interacting with Google services, like Maps) that you

can install. Uncheck the **Installed** checkbox. Then, if any of **Tools**, **Android 4.4 (API19)**, **Android 4.3 (API18)** and **Extras** appear in the **Packages** list, ensure that they're checked and click **Install # packages...** (# is the number of items to be installed) to display the **Choose Packages to Install** window. Most items in the **Extras** node are optional. For this book, you'll need the **Android Support Library** and **Google Play services**. The **Google USB Driver** is necessary for Windows users who wish to test apps on Android devices.]

6. In the **Choose Packages to Install** window, read the license agreements for each item. When you're done, click the **Accept License** radio button, then click the **Install** button. The status of the installation process will be displayed in the **Android SDK Manager** window.

Creating Android Virtual Devices (AVDs)

The **Android emulator**, included in the Android SDK, allows you to test apps on your computer rather than on an actual Android device. This is useful if you're learning Android and don't have access to Android devices, but can be *very* slow, so a real device is preferred if you have one. There are some hardware acceleration features that can improve emulator performance (developer.android.com/tools/devices/emulator.html#acceleration). Before running an app in the emulator, you must create an **Android Virtual Device (AVD)** which defines the characteristics of the device you want to test on, including the screen size in pixels, the pixel density, the physical size of the screen, 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. For this book, we use AVDs for Google's Android reference devices—the Nexus 4 phone, the Nexus 7 small tablet and Nexus 10 large tablet—which run unmodified versions of Android. To do so, perform the following steps:

1. Open Eclipse.
2. Select **Window > Android Virtual Device Manager** to display the **Android Virtual Device Manager** window, then select the **Device Definitions** tab (Fig. 2).

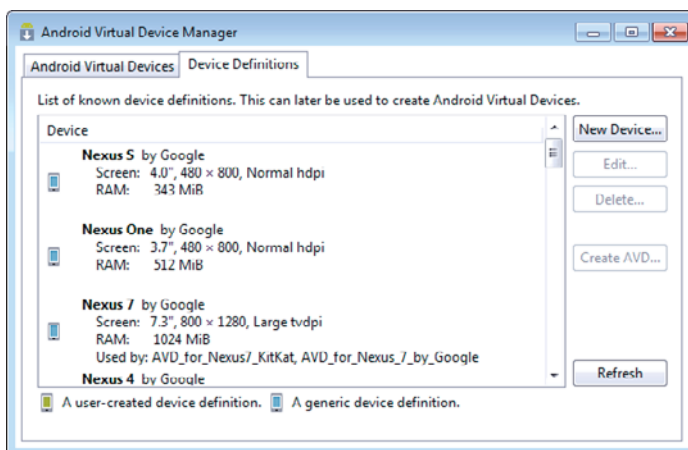


Fig. 2 | Android Virtual Device Manager window.

- Google provides preconfigured devices that you can use to create AVDs. Select **Nexus 4 by Google**, then click **Create AVD...** to display the **Create new Android Virtual Device (AVD)** window (Fig. 3), then configure the options as shown and click **OK** to create the AVD. If you check **Hardware keyboard present**, you'll be able to use your computer's keyboard to type data into apps that are running in the AVD, but this may prevent the soft keyboard from displaying on the screen. If your computer does not have a camera, you can select **Emulated** for the **Front Camera** and **Back Camera** options. Each AVD you create has many other options specified in its `config.ini`. You can modify this file as described at

<http://developer.android.com/tools/devices/managing-avds.html>
to more precisely match the hardware configuration of your device.

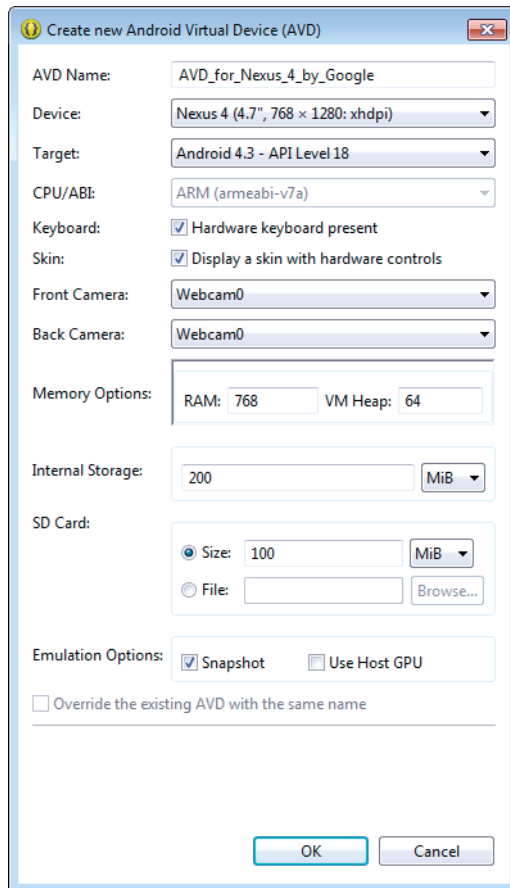


Fig. 3 | Configuring a Nexus 4 smartphone AVD for Android 4.3.

- We also configured Android 4.3 AVDs that represent Nexus 7 by Google and Nexus 10 by Google for testing our tablet apps. Their settings are shown in Fig. 4. In

addition, we configured Android 4.4 AVDs for the Nexus 4, Nexus 7 and Nexus 10 with the names: AVD_for_Nexus_4_KitKat, AVD_for_Nexus_7_KitKat, and AVD_for_Nexus_10_KitKat,

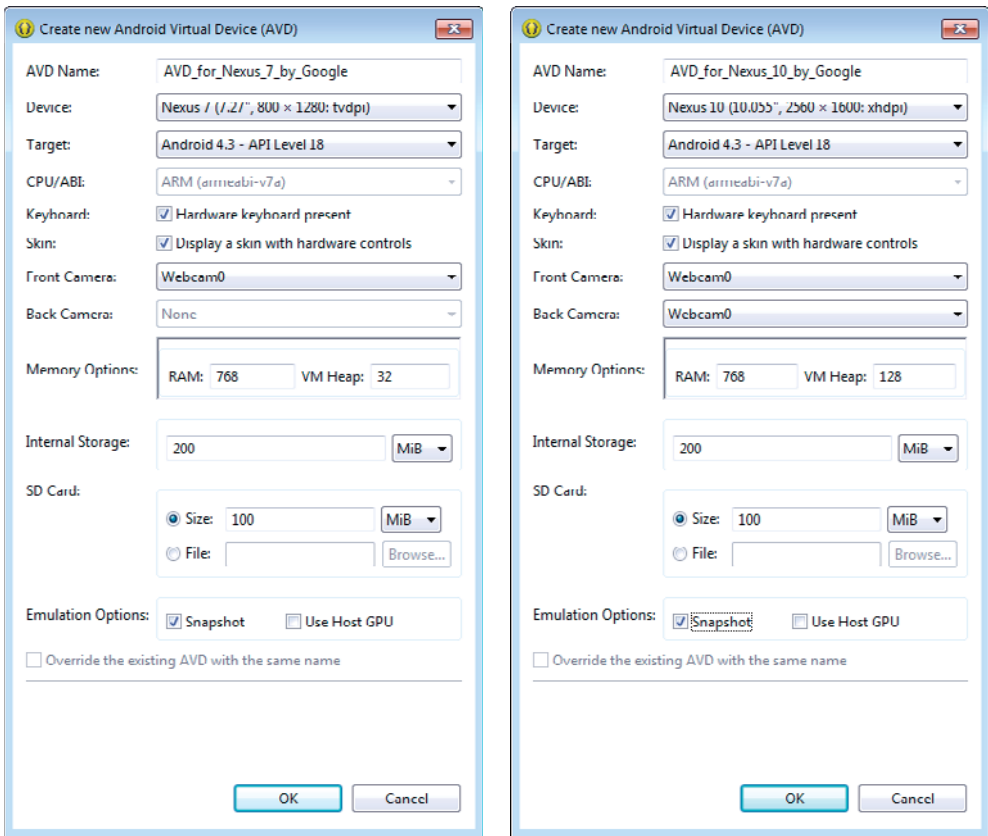


Fig. 4 | Configuring Nexus 7 and Nexus 10 tablet AVDs.

(Optional) Setting Up an Android Device for Development

As we mentioned, testing apps on AVDs can be slow due to AVD performance. If you have an Android device available to you, you should test the apps on that device. In addition, there are some features that 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. In some cases on Windows, you may also need 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>

Obtaining the Book's Code Examples

The examples for *Android for Programmers, 2/e, Volume 1* are available for download at

<http://www.deitel.com/books/AndroidFP2/>

If you're not already registered at our website, go to www.deitel.com and click the **Register** link. Fill in your information. Registration is free, and we do not share your information with anyone. Please verify that you entered your registration e-mail address correctly—you'll receive a confirmation e-mail with your verification code. *You must click the verification link in the e-mail before you can sign in at www.deitel.com for the first time.* Configure your e-mail client to allow e-mails from [deitel.com](http://www.deitel.com) to ensure that the verification e-mail is not filtered as junk mail. We send only occasional account-management e-mails unless you register separately for our free *Deitel® Buzz Online* e-mail newsletter at

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

Next, visit www.deitel.com and sign in using the **Login** link below our logo in the upper-left corner of the page. Go to <http://www.deitel.com/books/AndroidFP2/>. Click the **Examples** link to download a ZIP archive file containing the examples to your computer. Double click the ZIP file to unzip the archive, and make note of where you extract the file's contents on your system.

A Note Regarding the Android Development Tools

Google *frequently* updates the Android development tools. This often leads to problems compiling our apps when, in fact, the apps do not contain any errors. If you import one of our apps into Eclipse or Android Studio and it does not compile, there is probably a minor configuration issue. Please contact us by e-mail at deitel@deitel.com or by posting a question to:

- Facebook®—[facebook.com/DeitelFan](https://www.facebook.com/DeitelFan)
- Google+™—[google.com/+DeitelFan](https://plus.google.com/+DeitelFan)

and we'll help you resolve the issue.

You've now installed all the software and downloaded the code examples you'll need to study Android app development with *Android for Programmers, 2/e, Volume 1* and to begin developing your own apps. Enjoy!

This page intentionally left blank

3

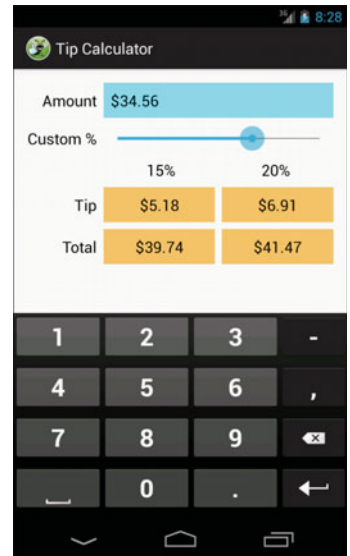
Tip Calculator App

Introducing GridLayout, LinearLayout, EditText, SeekBar, Event Handling, NumberFormat and Defining App Functionality with Java

Objectives

In this chapter you'll:

- Design a GUI using LinearLayouts and a GridLayout.
- Use the IDE's **Outline** window to add GUI components to LinearLayouts and a GridLayout.
- Use TextView, EditText and SeekBar GUI components.
- Use Java object-oriented programming capabilities, including classes, objects, interfaces, anonymous inner classes and inheritance to add functionality to an Android app.
- Programmatically interact with GUI components to change the text that they display.
- Use event handling to respond to user interactions with an EditText and a SeekBar.
- Specify that the keypad should always be displayed when an app is executing.
- Specify that an app supports only portrait orientation.



- 3.1 Introduction
- 3.2 Test-Driving the Tip Calculator App
- 3.3 Technologies Overview
 - 3.3.1 Class Activity
 - 3.3.2 Activity Lifecycle Methods
 - 3.3.3 Arranging Views with GridLayout and LinearLayout
 - 3.3.4 Creating and Customizing the GUI with the Graphical Layout Editor and the Outline and Properties Windows
 - 3.3.5 Formatting Numbers as Locale-Specific Currency and Percentage Strings
 - 3.3.6 Implementing Interface TextWatcher for Handling EditText Text Changes
- 3.4 Building the App's GUI
 - 3.4.1 GridLayout Introduction
 - 3.4.2 Creating the TipCalculator Project
 - 3.4.3 Changing to a GridLayout
 - 3.4.4 Adding the TextViews, EditText, SeekBar and LinearLayouts
 - 3.4.5 Customizing the Views to Complete the Design
- 3.5 Adding Functionality to the App
- 3.6 AndroidManifest.xml
- 3.7 Wrap-Up

3.1 Introduction

The **Tip Calculator** app (Fig. 3.1(a)) calculates and displays possible tips for a restaurant bill. As you enter each digit of a bill amount by touching the *numeric keypad*, the app calculates and displays the tip amount and total bill (bill amount + tip) for a 15% tip and a custom

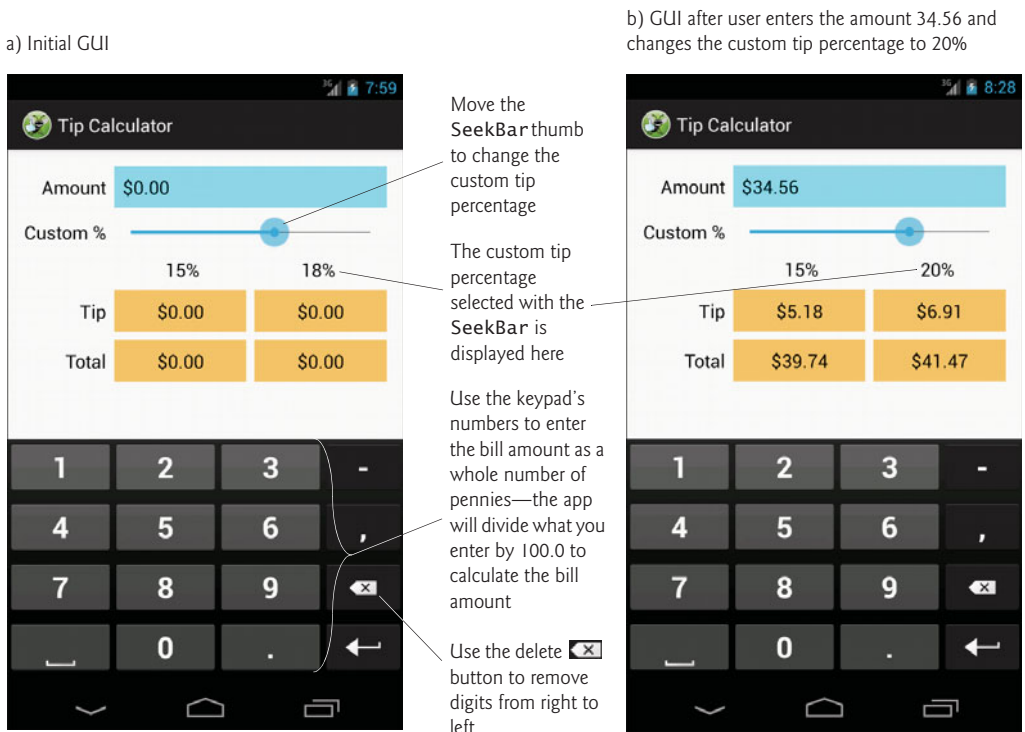


Fig. 3.1 | Entering the bill total and calculating the tip.

tip percentage (18% by default). You can specify a custom tip percentage from 0% to 30% by moving the `SeekBar thumb`—this updates the custom percentage shown and displays the custom tip and total (Fig. 3.1(b)). We chose 18% as the default custom percentage, because many restaurants in the United States add this tip percentage for parties of six people or more. The keypad in Fig. 3.1 may differ based on your AVD’s or device’s Android version, or based on whether you’ve installed and selected a custom keyboard on your device.

You’ll begin by test-driving the app—you’ll use it to calculate 15% and custom tips. Then we’ll overview the technologies you’ll use to create the app. You’ll build the app’s GUI using the Android Developer Tools IDE’s **Graphical Layout** editor and the **Outline** window. Finally, we’ll present the complete Java code for the app and do a detailed code walkthrough. We provide online an Android Studio version of Sections 3.2 and 3.4 at <http://www.deitel.com/books/AndroidFP2>.


3.2 Test-Driving the Tip Calculator App

Opening and Running the App

Open the Android Developer Tools IDE and import the **Tip Calculator** app project. Perform the following steps:

1. *Launching the Nexus 4 AVD.* For this test-drive, we’ll use the Nexus 4 smartphone AVD that you configured in the *Before You Begin* section. To launch the Nexus 4 AVD, select **Window > Android Virtual Device Manager** to display the **Android Virtual Device Manager** dialog. Select the Nexus 4 AVD and click **Start...**, then click the **Launch** button in the **Launch Options** dialog that appears.
2. *Opening the Import Dialog.* Select **File > Import...** to open the **Import** dialog.
3. *Importing the Tip Calculator app’s project.* Expand the **General** node, select **Existing Projects into Workspace**, then click **Next >** to proceed to the **Import Projects** step. Ensure that **Select root directory** is selected, then click **Browse...** In the **Browse For Folder** dialog, locate the `TipCalculator` folder in the book’s `examples` folder, select it and click **OK**. Ensure that **Copy projects into workspace** is *not* selected. Click **Finish** to import the project. It now appears in the **Package Explorer** window.
4. *Launching the Tip Calculator app.* Right click the `TipCalculator` project in the **Package Explorer** window, then select **Run As > Android Application** to execute **Tip Calculator** in the AVD.

Entering a Bill Total

Using the numeric keypad, enter 34.56. Just type 3456—the app will position the cents to the right of the decimal point. If you make a mistake, press the delete () button to erase one rightmost digit at a time. The `TextView`s under the **15%** and the custom tip percentage (**18%** by default) labels show the tip amount and the total bill for these tip percentages. All the **Tip** and **Total** `TextView`s update each time you enter or delete a digit.

Selecting a Custom Tip Percentage

Use the `SeekBar` to specify a *custom* tip percentage. Drag the `SeekBar`’s *thumb* until the custom percentage reads **20%** (Fig. 3.1(b)). As you drag the thumb, the tip and total for this custom tip percentage 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 features and Android technologies you'll use to build the **Tip Calculator** app. We assume that you're *already* familiar with Java object-oriented programming. You'll:

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

3.3.1 Class Activity

Unlike many Java apps, Android apps *don't have a main method*. Instead, they 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`). Users interact with an `Activity` through *views*—that is, GUI components. Before Android 3.0, a separate `Activity` was typically associated with each screen of an app. As you'll see, starting in Chapter 5, 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 often display multiple `Fragments` per screen to take better 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*. This is the `Activity` the user is interacting with.
- A *paused* `Activity` is *visible* on the screen but *does not* have the focus—such as when an alert dialog is displayed.
- A *stopped* activity is *not visible* on the screen and is likely to be killed by the system when its memory is needed. An `Activity` is *stopped* when another `Activity` becomes *active*.

As an `Activity` transitions among these states, the Android runtime calls various `Activity lifecycle methods`—all of which are defined in the `Activity` class

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

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 that the user 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.

3.3.3 Arranging Views with `LinearLayout` and `GridLayout`

Recall that layouts arrange views in a GUI. A `LinearLayout` (package `android.widget`) arranges views either *horizontally* (the default) or *vertically* and can size its views proportionally. We'll use this to arrange two `TextViews` horizontally and ensure that each uses half of the available horizontal space.

`GridLayout` (package `android.widget`) was introduced in Android 4.0 as a new layout for arranging views into cells in a rectangular grid. Cells can occupy *multiple* rows and columns, allowing for complex layouts. In many cases, `GridLayout` can be used to replace the older, and sometimes less efficient `TableLayout`, which arranges views into rows and columns where each row is typically defined as a `TableRow` and the number of columns is defined by the `TableRow` containing the most cells. Normally, `GridLayout` requires API level 14 or higher. However, the *Android Support Library* provides alternate versions of `GridLayout` and many other GUI features 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
```

A `GridLayout` *cannot* specify within a given row that the horizontal space should be allocated *proportionally* between multiple views. For this reason, several rows in this app's GUI will place two `TextViews` in a horizontal `LinearLayout`. This will enable you to place two `TextViews` in the same `GridLayout` cell and divide the cell's space evenly between them. 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
```

3.3.4 Creating and Customizing the GUI with the Graphical Layout Editor and the Outline and Properties Windows

You'll create `TextViews`, an `EditText` and a `SeekBar` using the IDE's **Graphical Layout** editor (that you used in Chapter 2) and **Outline** window, then customize them with the IDE's **Properties** window—which is displayed at the bottom of the **Outline** window when you're editing a GUI in the **Graphical Layout** editor. You'll do this *without* directly manipulating the XML stored in the files of the project's `res` folder.

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`**—often called a *slider* in other GUI technologies—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 custom tip percentage *only* from the more limited range 0 to 30.

In the **Properties** window, a view's most commonly customized properties typically appear at the top with their names displayed in bold (Fig. 3.2). All of a view's properties

are also organized into categories within the **Properties** window. For example, class `TextView` inherits many properties from class `View`, so the **Properties** window displays a **TextView** category with `TextView`-specific properties, followed by a **View** category with properties that are inherited from class `View`.

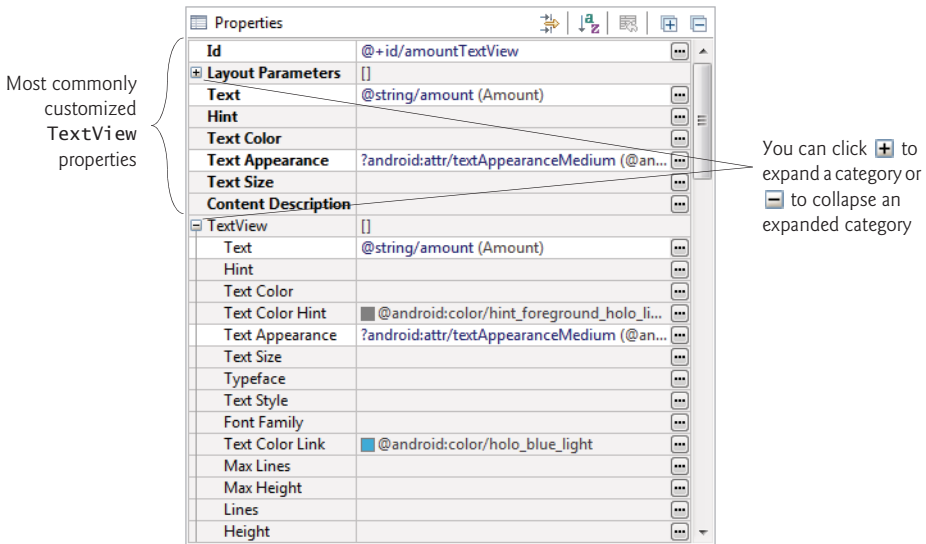


Fig. 3.2 | Properties window showing a `TextView`'s most commonly customized properties.

3.3.5 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 *internationalization*. You could also add *accessibility* strings and internationalize the app using the techniques you learned in Sections 2.7–2.8, though we did not do so in this app.

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

You'll use an *anonymous inner class* to implement the `TextWatcher` interface (from package `android.text`) to respond to *events when the user changes the text* in this app's `EditText`. 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.

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

You'll 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 custom tip percentage and to calculate the custom tip and total as the user moves the `SeekBar`'s thumb.

3.3.8 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 `New Android Application` dialog, such as the app's name, package name, target and minimum SDKs, Activity name(s), theme and more. You'll use the IDE's `Android Manifest` editor to add a new setting to the manifest that forces the *soft keyboard* to remain on the screen. You'll also specify that the app supports only *portrait orientation*—that is, the device's longer side is vertical.

3.4 Building the App's GUI

In this section, we'll show the precise steps for building the `Tip Calculator`'s GUI. The GUI will not look like the one shown in Fig. 3.1 until you've completed the steps. As you proceed through this section, the number of details presented may seem large, but they're repetitive and you'll get used to them as you use the IDE.

3.4.1 GridLayout Introduction

This app uses a `GridLayout` (Fig. 3.3) to arrange views into five rows and two columns. Each cell in a `GridLayout` can be *empty* or can hold one or more *views*, including layouts that *contain* other views. Views can span *multiple* rows or columns, though we did not use that capability in this GUI. You can specify a `GridLayout`'s number of rows and columns in the `Properties` window.

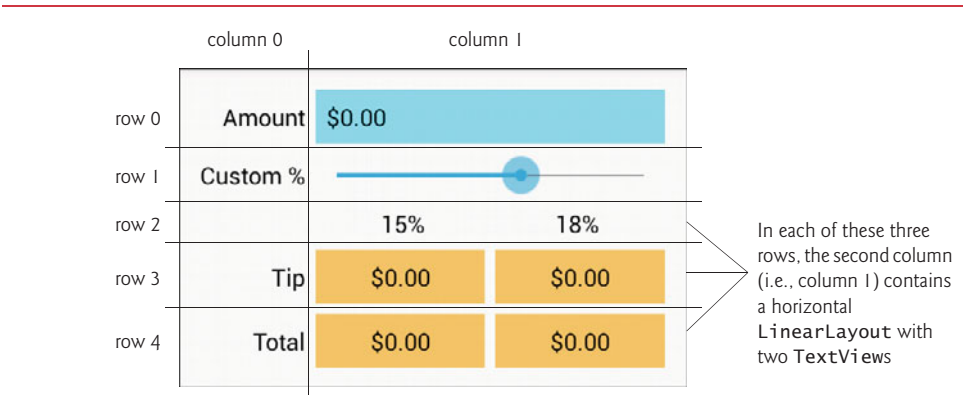


Fig. 3.3 | `Tip Calculator` GUI's `GridLayout` labeled by its rows and columns.

Each row's *height* is determined by the *tallest* view in that row. Similarly, the *width* of a column is defined by the *widest* view in that column. By default, views are added to a row from left to right. As you'll see, you can specify the exact row and column in which a view is to be placed. We'll discuss other `GridLayout` features as we present the GUI-building steps. To learn more about class `GridLayout`, visit:

<http://developer.android.com/reference/android/widget/GridLayout.html>

Id Property Values for This App's Views

Figure 3.4 shows the views' `id` property values. For clarity, our naming convention is to use the view's class name in the view's `id` property and Java variable name.

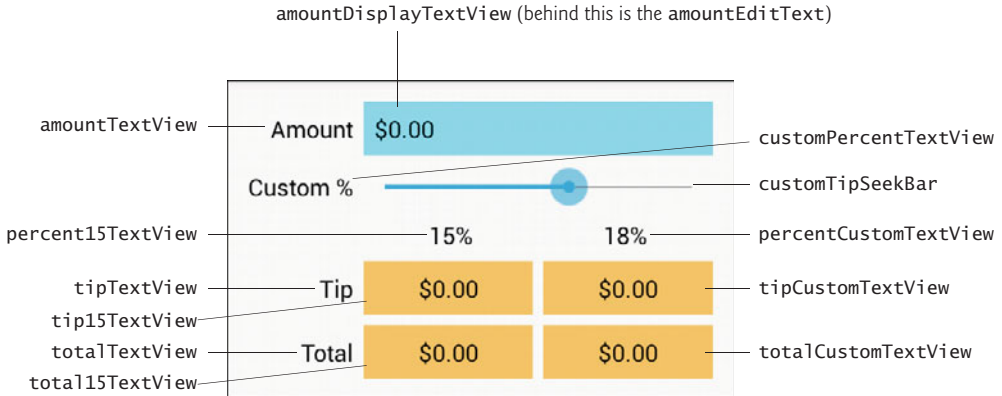


Fig. 3.4 | Tip Calculator GUI's components labeled with their `id` property values.

In the right column of the first row, there are actually *two* components in the *same* grid cell—the `amountDisplayTextView` is *hiding* the `amountEditText` that receives the user input. As you'll soon see, we restrict the user's input to integer digits so that the user cannot enter invalid input. However, we want the user to see the bill amount as a *currency* value. As the user enters each digit, we divide the amount by 100.0 and display the currency-formatted result in the `amountDisplayTextView`. In the *U.S. locale*, if the user enters 3456, as each digit is entered the `amountDisplayTextView` will show the values \$0.03, \$0.34, \$3.45 and \$34.56, respectively.

LinearLayout Id Property Values

Figure 3.5 shows the `ids` of the three horizontal `LinearLayout`s in the `GridLayout`'s right column.

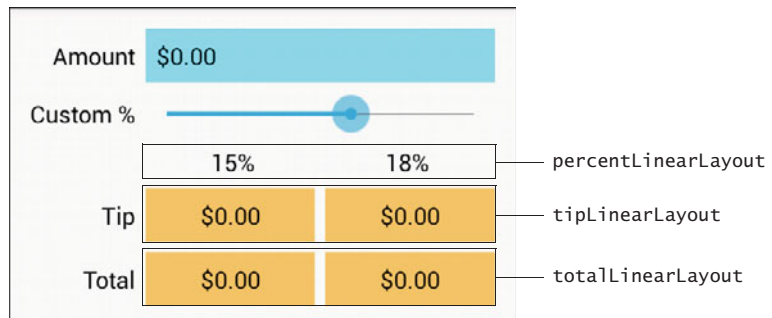


Fig. 3.5 | Tip Calculator GUI's `LinearLayout`s with their `id` property values.

3.4.2 Creating the TipCalculator Project

The Android Developer Tools IDE allows only *one* project with a given name per workspace, so before you create the new project, delete the TipCalculator project that you tested in Section 3.2. To do so, right click it and select **Delete**. In the dialog that appears, ensure that **Delete project contents on disk** is *not* selected, then click **OK**. This removes the project from the workspace, but leaves the project's folder and files on disk in case you'd like to look at our original app again later.

Creating a New Blank App Project

Next, create a new **Android Application Project**. Specify the following values in the **New Android Project** dialog's first **New Android Application** step, then press **Next** >:

- **Application Name:** Tip Calculator
- **Project Name:** TipCalculator
- **Package Name:** com.deitel.tipcalculator
- **Minimum Required SDK:** API18: Android 4.3
- **Target SDK:** API19: Android 4.4
- **Compile With:** API19: Android 4.4
- **Theme:** Holo Light with Dark Action Bar
- **Create Activity:** TipCalculator
- **Build Target:** Ensure that **Android 4.3** is checked

In the **New Android Project** dialog's second **New Android Application** step, leave the default settings, then press **Next** >. In the **Configure Launcher Icon** step, click the **Browse...** button, select the `DeitelGreen.png` app icon image (provided in the `images` folder with the book's examples) and click the **Open** button, then press **Next** >. In the **Create Activity** step, select **Blank Activity** (keep the default activity name), then press **Next** >. In the **Blank Activity** step, leave the default settings, then press **Finish** to create the project. Close `MainActivity.java` and `fragment_main.xml`, then open `activity_main.xml`. In the **Graphical Layout** editor, select **Nexus 4** from the screen-type drop-down list (as in Fig. 2.12). Once again, we'll use this device as the basis for our design.

3.4.3 Changing to a GridLayout

The default layout in `activity_main.xml` is a `FrameLayout`. Here, you'll change that to a `GridLayout`. Right click the `RelativeLayout` in the **Outline** window and select **Change Layout...** In the **Change Layout** dialog, select `GridLayout` and click **OK**. The IDE changes the layout and sets its `Id` to `GridLayout1`. We changed this to `gridLayout` using the `Id` field in the **Properties** window. By default, the `GridLayout`'s **Orientation** property is set to `horizontal`, indicating that its contents will be laid out row-by-row. Ensure that the `GridLayout`'s **Padding Left** and **Padding Right** properties are set to `activity_horizontal_margin` and that the **Padding Top** and **Padding Bottom** properties are set to `activity_vertical_margin`.

Specifying Two Columns and Default Margins for the GridLayout

Recall that the GUI in Fig. 3.3 consists of two columns. To specify this, select `gridLayout` in the **Outline** window, then change its **Column Count** property to 2 (in the **Properties** window's `GridLayout` group). By default, there are *no margins*—spaces that separate views—

around a `GridLayout`'s cells. Set the `GridLayout`'s **Use Default Margins** property to true to indicate that the `GridLayout` should place margins around its cells. By default, the `GridLayout` uses the recommended gap between views (8dp), as specified at

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

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

You'll now build the GUI in Fig. 3.3. 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. As you add each view to the GUI, immediately set its **Id** property using the names in Figs. 3.4–3.5. You can change the selected view's **Id** via the **Properties** window or by right-clicking the view (in the **Graphical Layout** editor or **Outline** window), selecting **Edit ID...** and changing the **Id** in the **Rename Resource** dialog that appears.

In the following steps, you'll use the **Outline** window to add views to the `GridLayout`. When working with layouts, it can be difficult to see the layout's *nested structure* and to place views in the correct locations by dragging them onto the **Graphical Layout** editor window. The **Outline** window makes these tasks easier because it shows the GUI's nested structure. Perform the following steps in the exact order specified—otherwise, the views will *not* appear in the correct order in each row. If this happens, you can reorder views by dragging them in the **Outline** window.

Step 1: Adding Views to the First Row

The first row consists of the `amountTextView` in the first column and the `amountEditText` behind the `amountDisplayTextView` in the second column. Each time you drop a view or layout onto the `GridLayout` in the **Outline** window, the view is placed in the layout's *next open cell*, unless you specify otherwise by setting the view's **Row** and **Column** properties. You'll do that in this step so that the `amountEditText` and `amountDisplayTextView` are placed in the same cell.

All of the `TextViews` in this app use the *medium*-sized font from the app's theme. The **Graphical Layout** editor's **Palette** provides *preconfigured* `TextViews` named **Large**, **Medium** and **Small** (in the **Form Widgets** section) to represent the theme's corresponding text sizes. In each case, the IDE configures the `TextView`'s **Text Appearance** property accordingly. Perform the following tasks to add the two `TextViews` and the `EditText`:

1. Drag a **Medium** `TextView` from the **Palette**'s **Form Widgets** section and drop it on the `GridLayout` in the **Outline** window. The IDE creates a new `TextView` named `textView1` and nests it in the `GridLayout` node. The default text "Medium Text" appears in the **Graphical Layout** editor. Change the `TextView`'s **Id** to `amountTextView`. You'll change its text in Step 6 (Section 3.4.5).
2. This app allows you to enter only *non-negative integers*, which the app divides by 100.0 to display the bill amount. The **Palette**'s **Text Fields** section provides many *preconfigured* `EditTexts` for various forms of input (e.g., numbers, times, dates, addresses and phone numbers). When the user interacts with an `EditText`, an appropriate keyboard is displayed based on the `EditText`'s *input type*. When you hover over an `EditText` in the **Palette**, a *tooltip* indicates the input type. From the **Palette**'s **Text Fields** section, drag a **Number** `EditText` (displayed with the number 42 on it) and drop it on the `GridLayout` node in the **Outline** window. Change the

`EditText`'s `Id` to `amountEditText`. The `EditText` is placed in the *second* column of the `GridLayout`'s *first* row.

3. Drag another **Medium** `TextView` onto the `GridLayout` node in the **Outline** window and change the `Id` to `amountDisplayTextView`. The new `TextView` is initially placed in the *first* column of the `GridLayout`'s *second* row. To place it in the *second* column of the `GridLayout`'s *first* row, set this `TextView`'s **Row** and **Column** properties (located in the **Properties** window's **Layout Parameters** section) to the values 0 and 1, respectively.

Step 2: Adding Views to the Second Row

Next, you'll add a `TextView` and `SeekBar` to the `GridLayout`. To do so:

1. Drag a **Medium** `TextView` (`customPercentTextView`) from the **Palette**'s **Form Widgets** section onto the `GridLayout` node in the **Outline** window.
2. Drag a `SeekBar` (`customTipSeekBar`) from the **Palette**'s **Form Widgets** section onto the `GridLayout` node in the **Outline** window.

Step 3: Adding Views to the Third Row

Next, you'll add a `LinearLayout` containing two `TextView`s to the `GridLayout`. To do so:

1. From the **Palette**'s **Layouts** section, drag a **Linear Layout (Horizontal)** (`percentLinearLayout`) onto the `GridLayout` node in the **Outline** window.
2. Drag a **Medium** `TextView` (`percent15TextView`) onto the `percentLinearLayout` node in the **Outline** window. This nests the new `TextView` in the `LinearLayout`.
3. Drag another **Medium** `TextView` (`percentCustomTextView`) onto the `percentLinearLayout` node in the **Outline** window.
4. The `percentLinearLayout` and its two nested `TextView`s should be placed in the second column of the `GridLayout`. To do so, select the `percentLinearLayout` in the **Outline** window, then set its **Column** property to 1.

Step 4: Adding Views to the Fourth Row

Next, you'll add a `TextView` and a `LinearLayout` containing two more `TextView`s to the `GridLayout`. To do so:

1. Drag a **Medium** `TextView` (`tipTextView`) onto the `GridLayout` node.
2. Drag a **Linear Layout (Horizontal)** (`tipLinearLayout`) onto the `GridLayout` node.
3. Drag two **Medium** `TextView`s (`tip15TextView` and `tipCustomTextView`) onto the `tipLinearLayout` node.

Step 5: Adding Views to the Fifth Row

To create the last row of the GUI, repeat Step 4, using the `Ids` `totalTextView`, `totalLinearLayout`, `total15TextView` and `totalCustomTextView`.

Reviewing the Layout So Far

The GUI and **Outline** window should now appear as shown in Fig. 3.6. The warning symbols shown in the **Graphical Layout** editor and the **Outline** window will go away as you complete the GUI design in Section 3.4.5.

a) GUI design so far

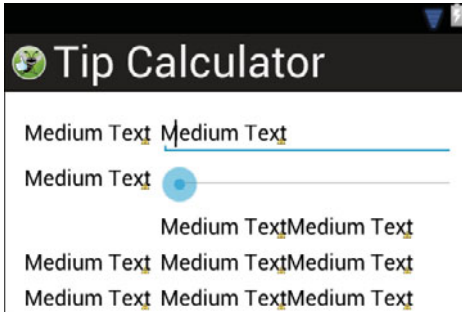
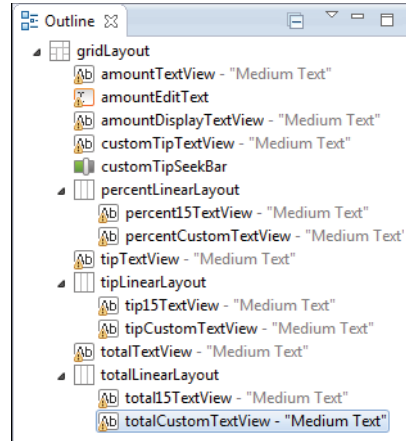
b) Outline window showing **Tip Calculator** components

Fig. 3.6 | The GUI and the IDE's **Outline** window after adding all the views to the GridLayout.

3.4.5 Customizing the Views to Complete the Design

You'll now complete the app's design by customizing the views' properties and creating several string and dimension resources. As you learned in Section 2.5, literal string values should be placed in the `strings.xml` resource file. Similarly, literal numeric values that specify view dimensions (e.g., widths, heights and spacing) should be placed in the `dimens.xml` resource file.

Step 6: Specifying Literal Text

Specify the literal text for the `amountTextView`, `customPercentTextView`, `percent15TextView`, `percentCustomTextView`, `tipTextView` and `totalTextView`:

1. Select the `amountTextView` in the **Outline** window.
2. In the **Properties** window, click the ellipsis button next to the **Text** property.
3. In the **Resource Chooser** Dialog, click **New String....**
4. In the **Create New Android String** dialog, specify **Amount** in the **String** field and **amount** in the **New R.string** field, then click **OK**.
5. In the **Resource Chooser** dialog, click **OK** to set the `amountTextView`'s **Text** property to the string resource identified as **amount**.

Repeat the preceding tasks for the other `TextView`s using the values shown in Fig. 3.7.

View	String	New R.string
<code>customPercentTextView</code>	Custom %	<code>custom_tip_percentage</code>
<code>percent15TextView</code>	15%	<code>fifteen_percent</code>

Fig. 3.7 | String resource values and resource IDs. (Part I of 2.)

View	String	New R.string
percentCustomTextView	18%	eighteen_percent
tipTextView	Tip	tip
totalTextView	Total	total

Fig. 3.7 | String resource values and resource IDs. (Part 2 of 2.)

Step 7: Right Aligning the TextViews in the Left Column

In Fig. 3.3, each of the left column's TextViews is right aligned. For the amountTextView, customPercentTextView, tipTextView and totalTextView, set the layout Gravity property to right—located in the Layout Parameters section in the Properties window.

Step 8: Configuring the amountTextView's Label For Property


Generally, each EditText should have a descriptive TextView that helps the user understand the EditText's purpose (also helpful for accessibility)—otherwise, *Android Lint* issues a warning. To fix this, you set the TextView's Label For property to the Id of the associated EditText. Select the amountTextView and set its Label For property (in the Properties window's View section) to

```
@+id/amountEditText
```

The + is required because the TextView is defined *before* the EditText in the GUI, so the EditText does not yet exist when Android converts the layout's XML into the GUI.

Step 9: Configuring the amountEditText

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

1. In the Properties window's Layout Parameters section, set the Width and Height to wrap_content. This indicates that the EditText should be just large enough to fit its content, including any padding.
2. Remove the layout Gravity value fill_horizontal, leaving the property's value blank. We'll discuss fill_horizontal in the next step.
3. Remove the Ems property's value, which indicates the EditText's width, measured in uppercase M characters of the view's font. In our GridLayout, this causes the second column to be too narrow, so we removed this default setting.
4. In the Properties window's TextView section, set Digits to 0123456789—this allows *only* digits to be entered, even though the numeric keypad contains minus (-), comma (,), period (.) and space buttons. By default, the Digits property is *not* displayed in the Properties window, because it's considered to be an advanced property. To display it, click the Show Advanced Properties () toggle button at the top of the Properties window.
5. We restricted the bill amount to a maximum of *six* digits—so the largest supported bill amount is 9999.99. In the Properties window's TextView section, set the Max Length property to 6.

Step 10: Configuring the `amountDisplayTextView`

To complete the formatting of the `amountDisplayTextView`, select it and set the following properties:

1. In the **Properties** window's **Layout Parameters** section, set the **Width** and **Height** to `wrap_content` to indicate that the `TextView` should be large enough to fit its content.
2. Remove the **Text** property's value—we'll programmatically display text here.
3. In the **Properties** window's **Layout Parameters** section, set the layout **Gravity** to `fill_horizontal`. This indicates that the `TextView` should occupy all remaining horizontal space in this `GridLayout` row.
4. In the **View** section, set the **Background** to `@android:color/holo_blue_bright`. This is one of several *predefined colors* (each starts with `@android:color`) in Android's *Holo* theme. As you start typing the **Background** property's value, a drop-down list of the theme's available colors is displayed. You can also use any *custom color* created from a combination of red, green and blue components called **RGB values**—each is an integer in the range 0–255 that defines the amount of red, green and blue in the color, respectively. Custom colors are defined in *hexadecimal (base 16) format*, so the RGB components are values in the range 00–FF. Android also supports *alpha (transparency)* values in the range 0 (*completely transparent*) to 255 (*completely opaque*). To use alpha, you specify the color in the format `#AARRGGBB`, where the first two hexadecimal digits represent the alpha value. If both digits of each color component are the same, you can use the abbreviated formats `#RGB` or `#ARGB`. For example, `#9AC` is treated as `#99AACC` and `#F9AC` is treated as `#FF99AACC`.
5. Finally, you'll add some padding around the `TextView`. To do so, you'll create a new *dimension resource* named `textview_padding`, which you'll use several times in the GUI. A view's **Padding** property specifies space on all sides of the view's content. In the **Properties** window's **View** section, click the **Padding** property's ellipsis button. Click **New Dimension...** to create a new *dimension resource*. Specify `textview_padding` for the **Name** and `8dp` for the **Value** and click **OK**, then select your new *dimension resource* and click **OK**.

Step 11: Configuring the `customPercentTextView`

Notice that the `customPercentTextView` is aligned with the top of the `customTipSeekBar`'s thumb. This looks better if it's *vertically centered*. To do this, in the **Properties** window's **Layout Parameters** section, modify the **Gravity** value from `right` to

```
right|center_vertical
```

The *vertical bar* (`|`) character is used to separate *multiple Gravity* values—in this case indicating that the `TextView` should be *right aligned* and *centered vertically* within the grid cell. Also set the `customPercentTextView`'s **Width** and **Height** properties to `wrap_content`.

Step 12: Configuring the `customTipSeekBar`

By default, a `SeekBar`'s range is 0 to 100 and its current value is indicated by its **Progress** property. This app allows custom tip percentages from 0 to 30 and specifies a default of 18. Set the `SeekBar`'s **Max** property to 30 and the **Progress** property to 18. Also, set the **Width** and **Height** to `wrap_content`.

Step 13: Configuring the `percent15TextView` and `percentCustomTextView`

Recall that `GridLayout` does *not* allow you to specify how a view should be sized relative to other views in a given row. This is why we placed the `percent15TextView` and `percentCustomTextView` in a `LinearLayout`, which *does* allow *proportional sizing*. A view's layout **Weight** (in certain layouts, such as `LinearLayout`) specifies the view's relative importance with respect to other views in the layout. By default, all views have a **Weight** of 0.

In this layout, we set **Weight** to 1 for `percent15TextView` and `percentCustomTextView`—this indicates that they have equal importance, so they should be sized equally. By default, when we added the `percentLinearLayout` to the `GridLayout`, its layout **Gravity** property was set to `fill_horizontal`, so the layout occupies the remaining space in the third row. When the `LinearLayout` is stretched to fill the rest of the row, the `TextView`s each occupy *half* of the `LinearLayout`'s width.

We also wanted each `TextView` to center its text. To do this, in the **Properties** window's **TextView** section, set the **Gravity** property to center. This specifies the `TextView`'s text alignment, whereas the *layout Gravity* property specifies how a view aligns with respect to the layout.

Step 14: Configuring the `tip15TextView`, `tipCustomTextView`, `total15TextView` and `totalCustomTextView`

To finalize these four `TextView`s, perform the following tasks on each:

1. Select the `TextView`.
2. Delete its **Text** value—we'll set this programmatically.
3. Set the **Background** to `@android:color/holo_orange_light`.
4. Set the layout **Gravity** to center.
5. Set the layout **Weight** to 1.
6. Set the layout **Width** to 0dp—this allows the layout to use the **Weight** to determine the view's width.
7. Set the `TextView` **Gravity** to center.
8. Set the `TextView` **Padding** to `@dimen/textview_padding` (the *dimension resource* you created in a previous step).

Notice that there's *no horizontal space* between the `TextView`s in the `tipLinearLayout` and `totalLinearLayout`. To fix this, you'll specify an 8dp right margin for the `tip15TextView` and `total15TextView`. In the **Properties** window's **Layout Parameters** section, expand the **Margin** section, then set the **Right** margin to 8dp by creating a new *dimension resource* named `textview_margin`. Next, use this resource to set the `total15TextView`'s **Right** margin.

Step 15: Vertically Centering the `tipTextView` and `totalTextView`

To vertically center the `tipTextView` and `totalTextView` with the other views in their respective rows, modify their layout **Gravity** properties from right to

```
right|center_vertical
```

When you do this for the `totalTextView`, the `GridLayout` centers this component vertically in the *remaining space from the fifth row to the bottom of the screen*. To fix this problem, drag a **Space** view (in the **Palette**'s **Layout** section) onto the `GridLayout` node in the **Outline**

window. This creates a sixth row that occupies the rest of the screen. As its name implies, a **Space** view occupies space in a GUI. The GUI should now appear as in Fig. 3.8.

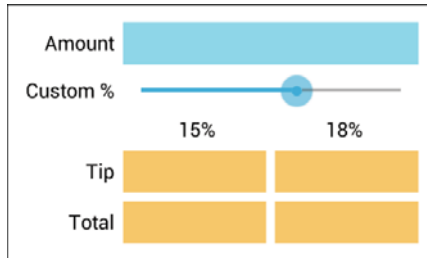


Fig. 3.8 | Final GUI design.

3.5 Adding Functionality to the App

Class `MainActivity` (Figs. 3.9–3.16) implements the **Tip Calculator** app’s functionality. It calculates the 15% and custom percentage tips and total bill amounts, and displays them in locale-specific currency format. To view the file, open `src/com.deitel.tipcalculator` and double click `MainActivity.java`. You’ll need to enter most of the code in Figs. 3.9–3.16.

The package and import Statements

Figure 3.9 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 bills using 15% and custom percentage tips.
3 package com.deitel.tipcalculator;
4
5 import java.text.NumberFormat; // for currency formatting
6
7 import android.app.Activity; // base class for activities
8 import android.os.Bundle; // for saving state information
9 import android.text.Editable; // for EditText event handling
10 import android.text.TextWatcher; // EditText listener
11 import android.widget.EditText; // for bill amount input
12 import android.widget.SeekBar; // for changing custom tip percentage
13 import android.widget.SeekBar.OnSeekBarChangeListener; // SeekBar listener
14 import android.widget.TextView; // for displaying text
15

```

Fig. 3.9 | `MainActivity`’s package and import statements.

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

- Class `NumberFormat` of package `java.text` (line 5) provides numeric formatting capabilities, such as *locale-specific* currency and percentage formats.

- Class `Activity` of package `android.app` (line 7) provides the basic *lifecycle methods* of an app—we’ll discuss these shortly.
- Class `Bundle` of package `android.os` (line 8) represents an app’s *state information*. Android gives an app the opportunity to *save its state* before another app appears on the screen. This might occur, for example, when the user *launches another app* or *receives a phone call*. The app that’s currently on the screen at a given time is in the *foreground* (the user can interact with it, and the app consumes the CPU) and all other apps are in the *background* (the user cannot interact with them, and they’re typically not consuming the CPU). When another app comes into the foreground, the app that was previously in the foreground is given the opportunity to *save its state* as it’s sent to the background.
- Interface `Editable` of package `android.text` (line 9) allows you to modify the content and markup of text in a GUI.
- You implement interface `TextWatcher` of package `android.text` (line 10) to respond to events when the user changes the text in an `EditText`.
- Package `android.widget` (lines 11–14) contains the *widgets* (i.e., views) and layouts that are used in Android GUIs. This app uses `EditText` (line 11), `SeekBar` (line 12) and `TextView` (line 14) widgets.
- You implement interface `SeekBar.OnSeekBarChangeListener` of package `android.widget` (line 13) to respond to the user moving the `SeekBar`’s *thumb*.

As you write code with various classes and interfaces, you can use the IDE’s **Source > Organize Imports** command to let the IDE insert the `import` statements for you. For cases in which the same class or interface name appears in more than one package, the IDE will let you select the appropriate `import` statement.

Tip Calculator App Activity and the Activity Lifecycle

Class `MainActivity` (Figs. 3.10–3.16) is the **Tip Calculator** app’s `Activity` subclass. When you created the `TipCalculator` project, the IDE generated this class as a subclass of `Activity` and provided an override of class `Activity`’s inherited `onCreate` method (Fig. 3.11). Every `Activity` subclass *must* override this method. The default code for class `MainActivity` also included an `onCreateOptionsMenu` method, which we removed because it’s not used in this app. We’ll discuss `onCreate` shortly.

```

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

```

Fig. 3.10 | Class `MainActivity` is a subclass of `Activity`.

Class Variables and Instance Variables

Lines 20–32 of Fig. 3.11 declare class `MainActivity`’s variables. The `NumberFormat` objects (lines 20–23) are used to format currency values and percentages, respectively. `NumberFormat` static method `getCurrencyInstance` returns a `NumberFormat` object that formats values as currency using the device’s *default locale*. Similarly, static method `getPercentInstance` formats values as percentages using the device’s *default locale*.

```

19 // currency and percent formatters
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 customPercent = 0.18; // initial custom tip percentage
27 private TextView amountDisplayTextView; // shows formatted bill amount
28 private TextView percentCustomTextView; // shows custom tip percentage
29 private TextView tip15TextView; // shows 15% tip
30 private TextView total15TextView; // shows total with 15% tip
31 private TextView tipCustomTextView; // shows custom tip amount
32 private TextView totalCustomTextView; // shows total with custom tip
33

```

Fig. 3.11 | 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 custom tip percentage (an integer in the range 0–30) that the user sets by moving the `SeekBar` *thumb* will be multiplied by 0.01 to create a `double` for use in calculations, then stored in `customPercent` (line 26). For example, if you select 25 with the `SeekBar`, `customPercent` will store 0.25, so the app will multiply the bill amount by 0.25 to calculate the 25% tip.

Line 27 declares the `TextView` that displays the currency-formatted bill amount. Line 28 declares the `TextView` that displays the custom tip percentage based on the `SeekBar` *thumb*'s position (see the 18% in Fig. 3.1(a)). The variables in line 29–32 will refer to the `TextView`s in which the app displays the calculated tips and totals.

Overriding Method onCreate of Class Activity

The `onCreate` method (Fig. 3.12)—which is *auto-generated* with lines 38–39 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 (Application Not Responding) dialog**—giving the user the option to *forcibly terminate the app*. You'll learn how to prevent this problem in Chapter 8.

```

34 // called when the activity is first created
35 @Override
36 protected void onCreate(Bundle savedInstanceState)
37 {
38     super.onCreate(savedInstanceState); // call superclass's version
39     setContentView(R.layout.activity_main); // inflate the GUI
40

```

Fig. 3.12 | Overriding `Activity` method `onCreate`. (Part I of 2.)

```

41     // get references to the TextViews
42     // that MainActivity interacts with programmatically
43     amountDisplayTextView =
44         (TextView) findViewById(R.id.amountDisplayTextView);
45     percentCustomTextView =
46         (TextView) findViewById(R.id.percentCustomTextView);
47     tip15TextView = (TextView) findViewById(R.id.tip15TextView);
48     total15TextView = (TextView) findViewById(R.id.total15TextView);
49     tipCustomTextView = (TextView) findViewById(R.id.tipCustomTextView);
50     totalCustomTextView =
51         (TextView) findViewById(R.id.totalCustomTextView);
52
53     // update GUI based on billAmount and customPercent
54     amountDisplayTextView.setText(
55         currencyFormat.format(billAmount));
56     updateStandard(); // update the 15% tip TextViews
57     updateCustom(); // update the custom tip TextViews
58
59     // set amountEditText's TextWatcher
60     EditText amountEditText =
61         (EditText) findViewById(R.id.amountEditText);
62     amountEditText.addTextChangedListener(amountEditTextWatcher);
63
64     // set customTipSeekBar's OnSeekBarChangeListener
65     SeekBar customTipSeekBar =
66         (SeekBar) findViewById(R.id.customTipSeekBar);
67     customTipSeekBar.setOnSeekBarChangeListener(customSeekBarListener);
68 } // end method onCreate
69

```

Fig. 3.12 | Overriding Activity method onCreate. (Part 2 of 2.)

onCreate's Bundle Parameter

During the app's execution, the user could change the device's configuration by *rotating the device* or *sliding out a hard keyboard*. For a good 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 38 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. You can find this class in your project's **gen folder**, which contains generated source-code files. 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 from your code (as we'll discuss momentarily). Some of the nested classes in class R include:

- class **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 **id**—contains constants for the *views* in your *XML layout files*
- class **layout**—contains constants that represent each *layout file* in your project (such as, `activity_main.xml`)
- class **string**—contains constants for each String in the `strings.xml` file.

Inflating the GUI

The call to `setContentview` (line 39) receives the constant `R.layout.activity_main` to indicate which XML file represents MainActivity's GUI—in this case, the constant represents the `main.xml` file. Method `setContentview` uses this constant to load the corresponding XML document, which is then parsed and converted 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 43–51 obtain references to the `TextViews` that are changed by the app. Lines 43–44 obtain a reference to the `amountDisplayTextView` that's updated when the user enters the bill amount. Lines 45–46 obtain a reference to the `percentCustomTextView` that's updated when the user changes the custom tip percentage. Lines 47–51 obtain references to the `TextViews` where the calculated tips and totals are displayed.

Displaying Initial Values in the TextViews

Lines 54–55 set `amountDisplayTextView`'s text to the initial `billAmount` (0.00) in a *locale-specific* currency format by calling the `currencyFormat` object's **format method**. Next, lines 56–57 call methods `updateStandard` (Fig. 3.13) and `updateCustom` (Fig. 3.14) to display initial values in the tip and total `TextViews`.

Registering the Event Listeners

Lines 60–61 get a reference to the `amountEditText`, and line 62 calls its `addTextChangedListener` method to register the `TextChangedListener` that will respond to *events* generated when the *user changes the text* in the `EditText`. We define this listener (Fig. 3.16) as an *anonymous-inner-class object* that's assigned to the instance variable `amountEditTextWatcher`.

Lines 65–66 get a reference to the `customTipSeekBar` and line 67 calls its `setOnSeekBarChangeListener` method to register the `OnSeekBarChangeListener` that will respond to *events* generated when the user moves the `customTipSeekBar`'s *thumb* to change the custom tip percentage. We define this listener (Fig. 3.15) as an *anonymous-inner-class object* that's assigned to the instance variable `customSeekBarListener`.

Method updateStandard of Class MainActivity

Method `updateStandard` (Fig. 3.13) updates the 15% tip and total `TextViews` each time the user *changes* the bill amount. The method uses the `billAmount` value to calculate the tip amount and the total of the bill amount and tip. Lines 78–79 display the amounts in currency format.

```

70 // updates 15% tip TextViews
71 private void updateStandard()
72 {
73     // calculate 15% tip and total
74     double fifteenPercentTip = billAmount * 0.15;
75     double fifteenPercentTotal = billAmount + fifteenPercentTip;
76
77     // display 15% tip and total formatted as currency
78     tip15TextView.setText(currencyFormat.format(fifteenPercentTip));
79     total15TextView.setText(currencyFormat.format(fifteenPercentTotal));
80 } // end method updateStandard
81

```

Fig. 3.13 | Method `updateStandard` calculates and displays the 15% tip and total.

Method `updateCustom` of Class `MainActivity`

Method `updateCustom` (Fig. 3.14) updates the custom tip and total `TextViews` based on the tip percentage the user selected with the `customTipSeekBar`. Line 86 sets the `percentCustomTextView`'s text to the `customPercent` value formatted as a percentage. Lines 89–90 calculate the `customTip` and `customTotal`. Then, lines 93–94 display the amounts in currency format.

```

82 // updates the custom tip and total TextViews
83 private void updateCustom()
84 {
85     // show customPercent in percentCustomTextView formatted as %
86     percentCustomTextView.setText(percentFormat.format(customPercent));
87
88     // calculate the custom tip and total
89     double customTip = billAmount * customPercent;
90     double customTotal = billAmount + customTip;
91
92     // display custom tip and total formatted as currency
93     tipCustomTextView.setText(currencyFormat.format(customTip));
94     totalCustomTextView.setText(currencyFormat.format(customTotal));
95 } // end method updateCustom
96

```

Fig. 3.14 | Method `updateCustom` calculates and displays the custom tip and total.

Anonymous Inner Class That Implements Interface `OnSeekBarChangeListener`

Lines 98–120 of Fig. 3.15 create the *anonymous-inner-class* object named `customSeekBarListener` that responds to `customTipSeekBar`'s *events*. If you're not familiar with *anonymous inner classes*, visit the following page:

<http://bit.ly/AnonymousInnerClasses>

Line 67 (Fig. 3.12) registered `customSeekBarListener` as `customTipSeekBar`'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.

```

97 // called when the user changes the position of SeekBar
98 private OnSeekBarChangeListener customSeekBarListener =
99     new OnSeekBarChangeListener()
100     {
101         // update customPercent, then call updateCustom
102         @Override
103         public void onProgressChanged(SeekBar seekBar, int progress,
104             boolean fromUser)
105         {
106             // sets customPercent to position of the SeekBar's thumb
107             customPercent = progress / 100.0;
108             updateCustom(); // update the custom tip TextViews
109         } // end method onProgressChanged
110
111         @Override
112         public void onStartTrackingTouch(SeekBar seekBar)
113         {
114         } // end method onStartTrackingTouch
115
116         @Override
117         public void onStopTrackingTouch(SeekBar seekBar)
118         {
119         } // end method onStopTrackingTouch
120     }; // end OnSeekBarChangeListener
121

```

Fig. 3.15 | Anonymous inner class that implements interface `OnSeekBarChangeListener` to respond to the events of the `customSeekBar`.

Overriding Method `onProgressChanged` of Interface `OnSeekBarChangeListener`

Lines 102–119 implement interface `OnSeekBarChangeListener`'s methods. Method `onProgressChanged` is called whenever the `SeekBar`'s *thumb* position *changes*. Line 107 calculates `customPercent` using the method's `progress` parameter—an `int` representing the `SeekBar`'s *thumb* position. We divide this by 100.0 to get the custom percentage. Line 108 calls method `updateCustom` to recalculate and display the custom 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 slider's thumb (`onStartTrackingTouch`) or *stops* moving it (`onStopTrackingTouch`), so we simply provide an *empty* body for each (lines 111–119) to *fulfill* the *interface contract*.

Anonymous Inner Class That Implements Interface `TextWatcher`

Lines 123–156 of Fig. 3.16 create the *anonymous-inner-class* object `amountEditTextWatcher` that responds to `amountEditText`'s *events*. Line 62 registered this object to *listen* for `amountEditText`'s events that occur when the text changes.

Overriding Method `onTextChanged` of Interface `TextWatcher`

The `onTextChanged` method (lines 126–144) is called whenever the text in the `amountEditText` is *modified*. The method receives four parameters. In this example, we use only

```

122 // event-handling object that responds to amountEditText's events
123 private TextWatcher amountEditTextWatcher = new TextWatcher()
124 {
125     // called when the user enters a number
126     @Override
127     public void onTextChanged(CharSequence s, int start,
128         int before, int count)
129     {
130         // convert amountEditText's text to a double
131         try
132         {
133             billAmount = Double.parseDouble(s.toString()) / 100.0;
134         } // end try
135         catch (NumberFormatException e)
136         {
137             billAmount = 0.0; // default if an exception occurs
138         } // end catch
139
140         // display currency formatted bill amount
141         amountDisplayTextView.setText(currencyFormat.format(billAmount));
142         updateStandard(); // update the 15% tip TextViews
143         updateCustom(); // update the custom tip TextViews
144     } // end method onTextChanged
145
146     @Override
147     public void afterTextChanged(Editable s)
148     {
149     } // end method afterTextChanged
150
151     @Override
152     public void beforeTextChanged(CharSequence s, int start, int count,
153         int after)
154     {
155     } // end method beforeTextChanged
156 }; // end amountEditTextWatcher
157 } // end class MainActivity

```

Fig. 3.16 | Anonymous inner class that implements interface `TextWatcher` to respond to the events of the `amountEditText`.

`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`.

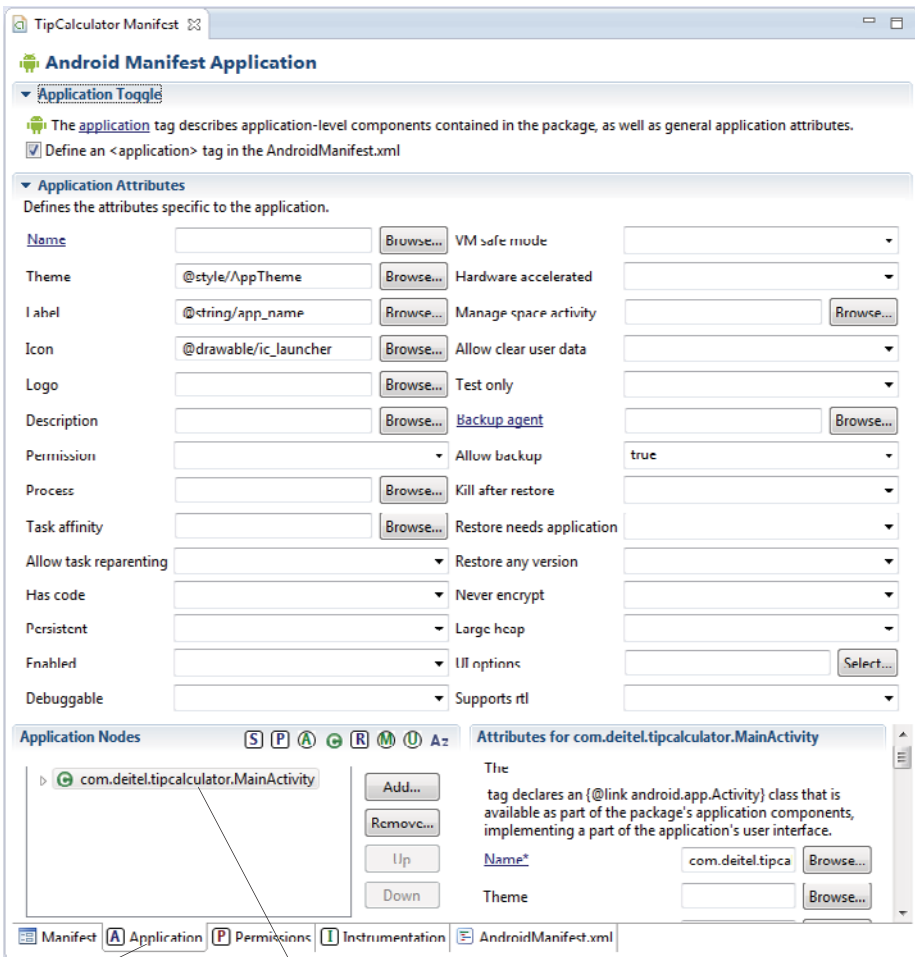
Line 133 converts the user input from `amountEditText` to a double. We allow users to enter only whole numbers in pennies, so we divide the converted value by 100.0 to get the actual bill amount—e.g., if the user enters 2495, the bill amount is 24.95. Lines 142–143 call `updateStandard` and `updateCustom` to recalculate and display the tips and totals.

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 146–155) to fulfill the interface contract.

3.6 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 *soft keypad* should *always* remain on the screen. You'll use the IDE's **Android Manifest** editor to specify these settings. To open the **Android Manifest** editor, double click the app's `AndroidManifest.xml` file in the **Package Explorer**. At the bottom of the editor, click the **Application** tab (Fig. 3.17), then select the `MainActivity` node in the **Application Nodes** section at the bottom of the window. This displays settings for the `MainActivity` in the **Attributes for com.deitel.tipcalculator.MainActivity** section.



Application tab Select this node to specify settings for the app's MainActivity

Fig. 3.17 | Android Manifest editor's Application tab.

Configuring MainActivity for Portrait Orientation

In general, most apps should support *both* portrait and landscape orientations. In *portrait* orientation, the device's height is greater than its width. In *landscape orientation*, the device's width is greater than its height. 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, you'll configure `MainActivity` to support *only* portrait orientation. In the **Android Manifest** editor's **Attributes for com.deitel.tipcalculator.MainActivity** section, scroll down to the **Screen orientation** option and select `portrait`.

Forcing the Soft Keypad to Always Display for MainActivity

In the **Tip Calculator** app, the soft keypad should be displayed immediately when the app executes and should remain on the screen at all times. In the **Android Manifest** editor's **Attributes for com.deitel.tipcalculator.MainActivity** section, scroll down to the **Window soft input mode** option and select `stateAlwaysVisible`. Note that this will *not* display the soft keyboard if a hard keyboard is present.

3.7 Wrap-Up

In this chapter, you created your first *interactive* Android app—the **Tip Calculator**. We overviewed the app's capabilities, then you test-drove it to calculate standard and custom tips based on the bill amount entered. You followed detailed step-by-step instructions to build the app's GUI using the Android Developer Tools IDE's **Graphical Layout** editor, **Outline** window and **Properties** window. We also walked through the code of the `Activity` subclass `MainActivity`, which defined the app's functionality.

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 that 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 changes the text in the `EditText`. You also defined an anonymous inner class that implements the `OnSeekBarChangeListener` interface so the app can calculate a new custom tip and total as the user changes the custom tip percentage by moving the `SeekBar`'s thumb.

Finally, you opened the `AndroidManifest.xml` file in the IDE's **Android Manifest** editor to specify that the `MainActivity` supports only portrait orientation and that the `MainActivity` should always display the keypad.

Using the IDE's **Graphical Layout** editor, **Outline** window, **Properties** window and **Android Manifest** editor enabled you to build this app without manipulating the XML in the project's resource files and `AndroidManifest.xml` file.

In the next chapter, we introduce collections while building the **Twitter® Searches** app. Many mobile apps display lists of items. You'll do this by using a `ListActivity` containing a `ListView` that's bound to an `ArrayList<String>`. You'll also store app data as user preferences and learn how to launch the device's web browser to display a web page.

Index

Symbols

(Android Developer Tools
rule markers in the
Graphical Layout editor
54

Numerics

100 Destinations 6

A

accelerometer **15**

listening 214

accelerometer sensor **201**, 215

access Android services 110

Accessibility

Content Description

property **58**, 105

Explore by Touch **37**, **57**

TalkBack **37**, **57**

TalkBack localization 62

accessibility 32, 37, 57, 103

explore-by-touch mode 9

Accessibility APIs 9

accessing Android content

providers 13

action bar 42, 126, 127

ACTION_SEND constant of class

Intent **121**

ACTION_VIEW constant of

class Intent **118**

Activity class **67**, 80

findFragmentById

method **133**, 150

getFragmentManager

method **133**, 150, 164

getMenuInflater method

151

getResources method **149**

getString method **118**

getString method with

multiple arguments 121

Activity class (cont.)

getService method 214

method 214

lifecycle methods 172

onCreate method **67**, 171

onCreateOptionsMenu

method **132**, 150

onDestroy method 171,

172

onOptionsItemSelected

method **132**, 151

onPause method 171, **172**

onResume method 171

onStart method **150**, 171

onStop method 171

runOnUiThread method

192

sent to background 177

setContentView method

83

setRequestedOrientation

method **149**

setVolumeControlStream

method **172**, 176

Activity Not Responding

(ANR) dialog 247

Activity templates 42

activity_main.xml **48**

ActivityNotFoundException

class 99

Adapter class **98**, 202

AdapterView class **98**, **111**

AdapterView.OnItemClickListener

interface **111**, 264

AdapterView.OnItemLongClick

Listener interface **111**

add a class to a project 175

add method of class

FragmentTransaction **257**

addCallback method of class

SurfaceHolder **181**

adding components to a row 70

addPreferencesFromResource

method of class

PreferenceFragment 165

Address Book app 15

addToBackStack method of

class FragmentTransaction

258

Adjust View Bounds property

of an ImageView 142

AdMob 296, 297

ADT (Android Development

Tools Plugin) 14

ADT Plugin for Eclipse 293

advertising revenue 297

AlertDialog class **99**, 110,

115, 201

AlertDialog.Builder class

99, 115

alpha (transparency) values 77

alpha animation for a View

146

alpha method of class Color

233

alternative-resource naming

conventions 59

Amazon Mobile app 297

analysis **18**

Android 2.2 (Froyo) 7

Android 2.3 (Gingerbread) **8**

Android 3.x

Honeycomb **8**

Android 4.0 (Ice Cream

Sandwich) **8**

Android APIs 5

Android app marketplaces 303

Amazon Appstore 303

AndroidPIT 303

AppTalism 303

GetJar 303

Handango 303

Moboro 303

Mplayit 303

- Android app marketplaces (cont.)
 - Opera Mobile Store 303
 - Samsung Apps 303
 - SlideMe 303
- Android Asset Studio 291
- Android Beam **9**, 10
- Android Cloud to Device Messaging (C2DM) **7**
- Android developer documentation (`developer.android.com`) xviii
- Android developer documentation (`developer.android.com/sdk/installing/studio.html`) xviii
- Android Developer Tools
 - Graphical Layout** editor **36**, 37, 45, 46, 48
- Android Developer Tools IDE 36, 37
- Android Development Tools (ADT) Plugin 14
- Android device manufacturers xv
- Android emulator **xxvi**, **14**, 37
 - Android for Programmers* page on InformIT xviii
 - Android for Programmers* website xv, xviii
- Android Jelly Bean **9**
- Android KitKat **10**
- Android Lint 39, 58
- Android Manifest** editor 70, **87**, **88**
- Android Market
 - language 301
 - location 302
 - price 302
- Android Newsgroups
 - Android Discuss 33
- Android project
 - res folder **45**, **50**
 - value folder **50**
- Android Resources** editor 60
- Android SDK xix, xxiii, xxvi, **2**, 14
- Android SDK 2.x xv, xvi
- Android SDK Manager** xxv
 - Android SDK versions and API levels 39
 - Android SDK/ADT Bundle xxiii
 - Android SDK/ADT bundle xxiv, xxv, 19, 38
 - Android services
 - access 110
 - Android source code and documentation
 - FAQs 4
 - governance philosophy 4
 - licenses 4
 - source code 4
 - Android Studio **3**, 13, **14**, 37
 - Android Support Library 68, 132, 203, **203**, 230
 - 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 Virtual Device (AVD) **xxvi**, **14**, 19, 23, 56
 - Setting hardware emulation options** 30
 - Android Virtual Device Manager** xxvi
 - `android:duration` attribute of a translate animation **147**
 - `android:fromXDelta` attribute of a translate animation **146**
 - `android:startOffset` attribute of a translate animation **147**
 - `android:toXDelta` attribute of a translate animation **146**
 - `android.app` package **67**, 80, 98, 110, 132, 133, 247
 - `android.content` package **98**, **110**, 202
 - `android.content.res` package 134, **149**, **157**
 - `android.database` package **247**
 - `android.database.sqlite` package **247**
 - `android.graphics` package 173, 202
 - `android.graphics.drawable` package 162
 - `android.media` package 172
 - `android.net` package **110**
 - `android.os` package 80, 135, **247**
 - `android.preference` package **132**
 - `android.text` package **69**, 80
 - `android.util` package 136, 180
 - `android.view` package **111**, 132, 172, 202
 - `android.view.animation` package 135
 - `android.view.inputmethod` package **111**
 - `android.widget` package **68**, 80, 98, 111, 135, 247
- Android@Home framework **9**
- AndroidLicenser 303
- `AndroidManifest.xml` **70**, 100
- anim folder of an Android project **46**, **134**
- animation xvii
 - alpha animation **146**
 - framework 8
 - manual 172
 - options in an XML file 135
 - rotate animation **146**
 - scale animation **146**
 - set **146**
 - thread 172
 - translate animation for a View **146**
 - tween **146**
 - View based **146**
- Animation class **135**
 - `setRepeatCount` method **136**, 157

- AnimationUtils class **135**, 157
 - loadAnimation method **135**, 157
- animator folder of an Android project **46**, **134**
- anonymous inner class 67
- ANR (activity not responding) dialog **81**, 113, 247, 172
- anti-aliasing 220
- .apk file (Android application package file) 292
- app xxiii
- app bar 24
- app development xxiii
- app platforms
 - Amazon Kindle 304
 - Android 304
 - BlackBerry 304
 - iPhone 304
 - Windows Mobile 304
- app review sites
 - Android and Me 306
 - Android App Review Source 306
 - Android Police 306
 - Android Tapp 306
 - AndroidGuys 306
 - AndroidLib 306
 - AndroidPIT 306
 - AndroidZoom 306
 - Androinica 306
 - AppBrain 306
 - Applicious 306
 - Appstorm 306
 - Best Android Apps Review 306
 - Phandroid 306
- app review video sites
 - Android Video Review 306
 - Applicious 306
 - Crazy Mike's Apps 306
 - Daily App Show 306
 - Life of Android 306
- app-driven approach xvi, **2**
- Application Not Responding (ANR) dialog **81**, 113
- application resource 13
 - application resources (developer.android.com/guide/topics/resources/index.html) **50**
 - apply method of class SharedPreferences.Edit or **117**
 - ARGB **231**
 - ARGB color scheme 25
 - argb method of class Color **234**
 - ARGB_8888 constant **222**
 - ArrayAdapter class **98**, **111**, 114, 247
 - ArrayList class 98, 110, **136**
 - asset 301
 - AssetManager class **134**
 - list method 158
 - assets folder of an Android app **133**
 - AsyncTask class **247**, 265, **266**, 267, 277, 278, 279, 280
 - execute method **265**
 - attribute
 - in the UML 18
 - of a class 16
 - of an object 18
 - AttributeSet class 180
 - audio xvii, 13
 - audio playback xix
 - audio recording xix
 - audio stream
 - music 181
 - audio streams 172
 - music 172
 - audio volume 172
 - AudioManager class **172**, 181
 - AVD (Android Virtual Device) **xxvi**, **14**, 19, 23
- B**
- back button 24
- back stack **246**, 258, 259, 261
 - pop 258
 - push **258**
- background
 - activity sent to 177
- Background** property of a view 77
- Bank of America app 297
- beginTransaction method of class FragmentManager **257**
- behavior
 - of a class 16
- Bezier curve 227
- bind data to a ListView 98
- Bitmap class **173**, **202**, 237
 - bitmap encoding **222**
 - createBitmap method **222**
 - eraseColor method **238**
- Bitmap.Config.ARGB_8888 constant **222**
- Blank Activity** template **42**
- blue method of class Color **233**
- Bluetooth Health Devices 9
- brand awareness 297
- branding apps
 - Amazon Mobile 297
 - Bank of America 297
 - Best Buy 297
 - CNN 297
 - Epicurious Recipe 297
 - ESPN ScoreCenter 297
 - NFL Mobile 297
 - NYTimes 297
 - Pocket Agent 297
 - Progressive Insurance 297
 - UPS Mobile 297
 - USA Today 297
 - Wells Fargo Mobile 297
 - Women's Health Workouts Lite 297
- Bundle class 80, **82**
 - for an Intent 122
 - putLong method **258**
- C**
- C2DM (Android Cloud to Device Messaging) 7
- Calendar API 9
- callback methods 246
- camera 5
- Cannon Game** app 15
- Canvas class **173**, 202
 - drawBitmap method **223**
 - drawCircle method **190**
 - drawLine method **191**
 - drawPath method **223**, 228

Canvas class (cont.)

drawRect method **190**drawText method **190**carrier billing **296**case-insensitive sort **114**cell in a `TableLayout` **70**

changeCursor method of class

`CursorAdapter` **267**characteristics of great apps **31**check-in **305**class **13, 17**instance variable **18**class library **5**

Classes

`Activity` **67, 80**`ActivityNotFoundException` **99**`Adapter` **98**`AdapterView` **98, 111**`AlertDialog` **99, 110**`AlertDialog.Builder` **99**`Animation` **135**`AnimationUtils` **135, 157**`ArrayAdapter` **98, 111, 114**`ArrayList` **98, 110, 136**`AssetManager` **134**`AsyncTask` **247, 265, 277**`AttributeSet` **180**`AudioManager` **172, 181**`Bitmap` **173, 202, 237**`Bundle` **80, 82**`Canvas` **173, 202**`Collections` **110, 136**`Color` **233**`Configuration` **149**`ContentResolver` **202**`ContentValues` **283**`Context` **110**`Cursor` **247**`CursorAdapter` **247, 264**`CursorFactory` **286**`DialogFragment` **132, 164**`DialogInterface` **110**`Display` **135, 150**`Drawable` **162**`EditText` **68, 80**`Fragment` **132**`FragmentManager` **133**

Classes (cont.)

`FragmentManager`**133, 246, 257, 258**`FrameLayout` **174**`GestureDetector.Simple``GestureListener` **224**`GestureDetector.Simple``OnGestureListener`**202**`GridLayout` **68, 101**`Handler` **135**`ImageButton` **98, 104, 111**`ImageView` **37, 54**`InputMethodManager` **111**`InputStream` **162**`Intent` **99, 110**`LayoutInflater` **133**`LinearLayout` **68**`ListActivity` **98, 110**`ListFragment` **247, 248**`ListPreference` **133**`ListView` **98**`Log` **136, 159**`MediaStore` **202**`MediaStore.Images.Media``data` **202**`Menu` **132, 150**`MenuInflater` **151, 267**`MotionEvent` **172, 194,****202, 226**`MultiSelectListPreference``133``NumberFormat` **69, 79**`Paint` **173**`Path` **202**`Preference` **133**`PreferenceFragment` **132,****165**`PreferenceManager` **133,****149**`PrintHelper` **230**`R` **82**`R.drawable` **82**`R.id` **83**`R.layout` **83**`R.string` **83**`Resources` **149, 157**`ScrollView` **251**`SeekBar` **66, 68, 80**`Sensor` **201**

Classes (cont.)

`SensorEvent` **216**`SensorManager` **214**`SharedPreferences` **98,****110, 111**`SharedPreferences.Editor``98, 117``SimpleCursorAdapter`**264**`SoundPool` **172, 181**`SQLiteDatabase` **247**`SQLiteOpenHelper` **247**`SurfaceHolder` **173, 181**`SurfaceView` **173, 181**`TableLayout` **70**`TextView` **37, 50, 68, 80**`Thread` **172, 195**`Toast` **135, 153**`Uri` **110, 119**`View` **111, 173**`ViewGroup` **251**`WindowManager` **135, 150**client area **36, 98**close method of class `Cursor`**267**

close method of class

`SQLiteOpenHelper` **283**cloud computing **7**code file **301**code highlighting **xviii, 2**code license **xv**code walkthrough **2**

collection

shuffle **162**`Collections` class **110, 136**shuffle method **136**sort method **114**collision detection **173, 184,****186**color **173**`Color` class **233**alpha method **233**argb method **234**blue method **233**green method **233**red method **233**

color folder of an Android

project **46, 134**`colors.xml` **139**

- Column Count property of a GridLayout 72
 - Column property of a LinearLayout 74
 - commit method of class FragmentTransaction **257**
 - Comparator<String> object String.CASE_INSENSITIVE_ORDER 114
 - compiling apps 290
 - component 16
 - Configuration class **149**
 - Constants
 - MODE_PRIVATE **113**
 - MODE_WORLD_READABLE **113**
 - MODE_WORLD_WRITEABLE **113**
 - contain other Views **251**
 - Content Description property **58**, 105
 - ContentResolver class **202**
 - ContentValues class **283**
 - Context class **110**
 - getSharedPreferences method **113**
 - startActivity method **99**, 119
 - ContextWrapper class
 - getAssets method **158**, 162
 - control 15
 - corners element of a shape **250**
 - crash report 303
 - Create New Android String** dialog 51
 - createBitmap method of class Bitmap **222**
 - createChooser method of class Intent **122**
 - createFromStream method of class Drawable **162**
 - creating a database 282
 - cryptographic key 290
 - CT
 - Google Play and App Business Issues 289
 - Cursor class **247**, 280, 285
 - close method **267**
 - Cursor class (cont.)
 - getColumnIndex method **280**
 - getColumnIndexOrThrow method **280**
 - getString method **280**
 - moveToFirst method **280**
 - CursorAdapter class **247**, 264
 - changeCursor method **267**, **267**
 - getCursor method **267**
 - CursorFactory class 286
 - custom subclass of View 178
 - custom view 171
- D**
- Dalvik Debug Monitor Service (DDMS) 293
 - data binding 98
 - database
 - creating 282
 - opening 282
 - upgrading 282
 - version number 286
 - Daydream 10
 - DDMS (Dalvik Debug Monitor Server) 293
 - DDMS perspective
 - LogCat** tab **136**
 - debugging
 - logging exceptions **136**, 159
 - default preferences 149
 - default resources 59
 - Deitel Facebook page 305
 - Deitel Web site xxix
 - Deitel® Buzz Online Newsletter xxix, 309
 - Deitel® Training (www.deitel.com/training) 309
 - delete method of class SQLiteDatabase **285**
 - density-independent pixels dp **52**
 - design process **18**, 18
 - Dev Guide 290
 - developer documentation
 - Keeping Your App Responsive* 33
 - Launch Checklist* 291
 - developer documentation (cont.)
 - Performance Tips* 33
 - Signing Your Applications* 293
 - Tablet App Quality Checklist* 291
 - Developer options 10
 - developer registration 299
 - device configuration 13
 - Device Screen Capture** window 294
 - DialogFragment class **132**, 164
 - onCreateDialog method **164**
 - show method **164**
 - DialogInterface class **110**
 - DialogInterface.OnClickListener interface **110**
 - digital certificate **293**
 - digitally sign your app 293
 - Digits** property of an EditText 76
 - dimens.xml 103
 - dimension resource 103
 - disabilities 37, 57
 - Display class **135**, 150
 - documentation
 - Android Design* 32
 - App Components* 32
 - application resources (developer.android.com/guide/topics/resources/index.html) **50**
 - Class Index* 32
 - Data Backup* 32
 - Debugging* 33
 - Get Started with Publishing* 33
 - Getting Started with Android Studio* 33
 - Google Play Developer Distribution Agreement* 33
 - Launch Checklist (for Google Play)* 33
 - Managing Projects from Eclipse with ADT* 33
 - Managing Your App's Memory* 33
 - Package Index* 32

documentation (cont.)

Security Tips 32

Tools Help 33

Using the Android Emulator
32

doInBackground method of
class AsyncTask 265, **266**,
267, 278

Doodlz app **19**

downloading source code xviii

dp (density-independent pixels)
52

drag event 227

draw

circles 173

lines 173

text 173

Drawable class **162**

createFromStream
method **162**

drawable folder of an Android
project **46**

Drawable resource

shape element **250**

drawBitmap method of class
Canvas **223**

drawCircle method of class
Canvas **190**

drawing characteristics 173

color 173

font size 173

line thickness 173

drawLine method of class
Canvas **191**

drawPath method of class
Canvas **223**, 228

drawRect method of class
Canvas **190**

drawText method of class
Canvas **190**

drive sales 297

E

e method of class Log **159**

Eclipse xix

import project 66, 91, 128,
171, 245

Outline window 66, 68

Eclipse documentation
(www.eclipse.org/
documentation) xviii

Eclipse IDE **2**

edit method of class

SharedPreferences **117**

Editable interface 80

EditText

Digits property 76

Ems property 76

Max Length property 76

EditText class **68**, 80

Hint property **103**, 105

IME Options property **103**,
105

input type 73

restrict maximum number of
digits 68

Ems property of an EditText
76

emulator 14, 291

gestures 15

emulator functionality 15

emulator gestures and controls
15

encapsulation **18**, 18

End User License Agreement
(EULA) 290, **291**

eraseColor method of class
Bitmap **238**

event handler

returning false **224**

event handling 67

events **5**

Examples xxix

execSQL method of class

SQLiteDatabase **287**

execute method of class

AsyncTask **265**

explicit Intent **99**, **136**, 151

Explore by Touch **37**, **57**

F

face detection 9

Facebook 96, **305**

Deitel page 305

file system access 13

final local variable for use in
an anonymous inner class 120

financial transaction 299

findFragmentById method of
class Activity **133**, 150

fling touch event 202

Folders

res/raw **171**, 175

folders

assets **133**

res/drawable-mdpi 250

font size 173

format method of class

NumberFormat **83**

format specifier

multiple in a String

resource 137

numbering in a String

resource 137

formatting strings 137

forums 33

Android Forums 33

Stack Overflow 33

fragment 8, **132**

Fragment class 67, **132**

getActivity method 157

getResources method **157**

onActivityCreated

method 176

onAttach method **201**,

233, 263, 270, 275

onCreate method 165

onCreate method **133**

onCreateOptionsMenu

method **217**

onCreateView method

133, 155

onCreateView method 176

onDestroy method **172**,

177

onDetach method **201**,

233, 263, 270, 275

onOptionsItemSelected

method **217**

onPause lifecycle method

215

onPause method **172**, 177

onResume method 265, 277

onSaveInstanceState

method **246**, 277

onStart lifecycle method

214

onStop method **267**

- Fragment class (cont.)
 - onViewCreated method **263**
 - setArguments method **258**
 - setRetainInstance method **264**
 - Fragment layout 140
 - Fragment lifecycle 201, 263, 265, 267, 270, 275, 277
 - fragment lifecycle 133
 - Fragment lifecycle methods 233
 - FragmentManager class **133**
 - beginTransaction method **257**
 - getFragmentByTag method **164**
 - popBackStack method **258**
 - FragmentTransaction class **133, 246, 257, 258**
 - add method **257**
 - addToBackStack method **258**
 - commit method **257**
 - replace method **258**
 - FrameLayout class **174**
 - fraudulent order 300
 - free app 295
 - Froyo (Android 2.2) **7**
 - Fullscreen Activity** template **42**
 - fully qualify a custom View's class name in an XML layout 171
 - future proof 32
- G**
- game loop **172, 183, 184, 195**
 - games 31
 - gaming console 5
 - gen folder of an android project **82**
 - gesture 5
 - double tap 5
 - double touch 5
 - drag 5
 - long press 5
 - pinch zoom 5
 - Swipe 5
 - touch 5
 - GestureDetector.OnDoubleTapListener interface **202, 224**
 - GestureDetector.OnGestureListener interface **202**
 - GestureDetector.SimpleGestureListener class **202, 224**
 - GestureDetector.SimpleGestureListener>default para font> class onSingleTap method **224**
 - Gestures
 - drag 5
 - long press 5
 - pinch 5
 - tap 5
 - getActionIndex method of class MotionEvent **226**
 - getActionMasked method of class MotionEvent **226**
 - getActivity method of class Fragment 157
 - getAll method of class SharedPreferences **113**
 - getAssets method of class ContextWrapper **158, 162**
 - getColumnIndex method of class Cursor **280**
 - getColumnIndexOrThrow method of class Cursor **280**
 - getConfiguration method of class Resources **149**
 - getCursor method of class CursorAdapter **267**
 - getDefaultSensor method of class SensorManager **214**
 - getFragmentByTag method of class FragmentManager **164**
 - getFragmentManager method of class Activity **133, 150, 164**
 - getHolder method of class SurfaceView **181**
 - getItemID method of class MenuItem **218**
 - getListView method of class ListViewFragment **264**
 - getListViewDefault Para Font> method of class ListActivity **114**
 - getMenuInflater method of class Activity **151**
 - getPointerCount method of class MotionEvent **227**
 - getResources method of class Activity **149**
 - getResources method of class Fragment **157**
 - getSharedPreferences method of class Context **113**
 - getString method of class Activity **118, 121**
 - getString method of class Cursor **280**
 - getString method of class Resources **157**
 - getString method of class SharedPreferences **118**
 - getStringSet method of class SharedPreferences **153**
 - getSystemService method of class Activity 214
 - getSystemUiVisibility method of class View **224**
 - getWritableDatabase method of class SQLiteOpenHelper **282**
 - getX method of class MotionEvent 227
 - getY method of class MotionEvent 227
 - Google APIs 5
 - Google Cloud Messaging 7
 - Google Maps 6
 - Google Play **11, 290, 291, 296, 299, 306**
 - countries 302
 - crash report 303
 - fees 300
 - high-resolution app icon 301
 - promotional graphic 301
 - promotional video 294, 301
 - publish 300, 301
 - Publish an Android App on Google Play** 301
 - publisher account 298
 - screenshots 301

Google Play Developer Console 303
Google Play Developer Program Policies 299
 Google Play game services xix
 Google Wallet 290, 296, **300**
 merchant account 302
 Google+ 96
 GPS xix
Graphical Layout editor 66
Graphical Layout editor in the
 Android Developer Tools **36**,
 37, 45, 46, 48
 graphics xvii, 13
Gravity property (layout) 76
Gravity property of a
 component **53**
 gravity sensor 201
 green method of class `Color`
 233
 GridLayout
 Column Count property 72
 Orientation property 72
 Use Default Margins
 property 73
 GridLayout class **68, 101**
 documentation 70
 gesture 15
 GUI components
 EditText **68**
 ImageButton **98, 104, 111**
 ImageView **37, 54**
 naming convention 71
 programmatically create 133
 ScrollView **251**
 SeekBar **66, 68**
 TextView **37, 47, 50**
 ViewGroup **251**
 GUI components are not thread
 safe 135
 GUI design 31
 GUI thread 247
 gyroscope sensor 201

H

Handler class **135**
 postDelayed method **135**,
 164
 hardware support 13
 hashtag 305

height of a table row 70
 hide the soft keyboard 114
 hint in an EditText 253
 Hint property of an EditText
 103, 105
 Holo Dark theme 40
 Holo Light theme 40
 Holo Light with dark action
 bars theme 40
 Holo user interface 8, 9
 home button 24
 HTML5 mobile apps xix

I

i-Newswire 307
 icon 290, **291**
 icon design firms
 99designs 292
 Aha-Soft 292
 Androidicons 292
 E lance 292
 glyphlab 292
 Iconiza 292
Id property of a layout or
 component **49**
 IDE (integrated development
 environment) 14
 ImageButton class **98, 104**,
 111
 images xvii
 ImageView class **37, 54**
 Adjust View Bounds
 property 142
 Scale Type property 142
IME Options 253
 IME Options property of an
 EditText **103, 105**
 immersive mode 24, 199, **202**,
 223, 224
 implicit Intent **99**
 import an existing project into
 Eclipse 66, 91, 128, 171, 245
Import dialog 21, 66, 171, 245
 in-app advertising 295, **297**
 in-app purchase 295
 in-app billing 298
 security best practices 298
 in-app purchase 298
 inflate method of class
 LayoutInflater **155**
 inflate method of class
 MenuInflater **151**
 inflate the GUI 182
 inflating a GUI **83**
 information hiding **18**
 inheritance **18**
 Input Type 253
 input type of an EditText 73
 InputMethodManager class
 111
 InputStream class 162
 setImageDrawable
 method **162**
 insert method of class
 SQLiteDatabase **283**
 insertImage method of class
 MediaStore.Images.Medi
 a **202**
 instance **17**
 instance variable **18**
 integrated development
 environment (IDE) 14
 intent chooser 96, **99**
 Intent class **99, 110**
 ACTION_SEND constant **121**
 ACTION_VIEW constant **118**
 Bundle 122
 createChooser method
 122
 explicit **99, 136**
 implicit **99**
 putExtra method **122**
 intent extras 122
 intent filter 99
 intent messaging 99
 interface
 implementing methods in
 Java **85**
 Interfaces
 AdapterView.
 OnItemClickListener
 111, 264
 AdapterView.OnItem-
 LongClickListener
 111
 DialogInterface.
 OnClickListener **110**
 Editable 80

Interfaces (cont.)

- GestureDetector.
 - OnDoubleTapListener
 - 202, 224
 - GestureDetector.
 - OnGestureListener
 - 202
 - List **136**
 - OnSeekBarChangeListener
 - Listener 84
 - Runnable 135
 - SeekBar.OnSeekBarChangeListener 69, 80, 234
 - SensorEventListener 215
 - Set **136**
 - SurfaceHolder.Callback
 - 173, 181, 193
 - TextWatcher **69**, 80
 - View.OnClickListener
 - 111

internationalization **37**, **59**, 59, 69

Internet public relations resources

- ClickPress 307
- i-Newswire 307
- Marketwire 307
- Mobility PR 307
- openPR 307
- PR Leap 307
- Press Release Writing 307
- PRLog 307
- PRWeb 307

invalidate method of class View **222**

J

- J2ObjC 304
- Java xvi, 5
- Java code xxiii
- Java developer documentation (www.oracle.com/technetwork/java/javase/downloads/index.html) xviii
- Java for Programmers, 2/e* (www.deitel.com/books/JavaFP2/) xvi

Java Fundamentals: Parts I and II (www.deitel.com/books/LiveLessons/) xvi

Java How to Program (www.deitel.com/books/jhtp10/) xvi

Java SE 7 Software Development Kit xxiii

- java.io package 162
- java.text package **69**, 79
- java.util package 136

K

- key/value pairs
 - persistent 110
- keyboard 5
- keyboard types 253
- keySet method of interface Map **113**
- key-value pairs associated with an app 98

L

- label 290
- Label For** property of a TextView 76
- landscape orientation 57, 88
- large-screen device 8
- layout 13
- layout folder of an Android project **46**
- LayoutInflater class **133**
 - inflate method **155**
- Layouts
 - GridLayout **68**
 - LinearLayout **68**
- layouts
 - activity_main.xml **48**
 - GridLayout **101**
 - RelativeLayout **46**
 - TableLayout **70**
- license for Android 4
- licensing policy 292
- licensing service **292**
- lifecycle methods 172
- lifecycle methods of an app 80
- light sensor 201
- line thickness 173
- linear acceleration sensor 201

LinearLayout class **68**

- Column** property 74
- linking your apps 302
- Linux 14
- List interface **136**
- list method of class
 - AssetManager 158
- ListActivity class **98**, 110
 - custom GUI 98
 - getListView method **114**
 - setListAdapter method **114**
- ListFragment class **247**, 248, 261
 - built-in ListView 263
 - getListView method **264**
 - setEmptyText method **264**
 - setListAdapter method **265**
- ListPreference class **133**
- ListView
 - data binding 98
- ListView class **98**, 261
 - format of a list item 108
 - setChoiceMode method **264**
- load a URL into a web browser 99
- load method of class
 - SoundPool **182**
- loadAnimation method of class AnimationUtils **135**, 157
- localization 50, 59, 137
- Localization Checklist 62
- localized resources 59
- lock screen widgets 10
- lockCanvas method of class SurfaceHolder **196**
- Log class **136**, 159
 - e method **159**
- LogCat** tab in the Android DDMS perspective **136**
- logcat tool **136**
- logging exceptions **136**, 159
- long press 94
- long-press touch event 202
- long-running operations 247

M

- Mac OS X 14
 - magnetic field sensor 201
 - makeText method of class Toast **153**
 - manifest file 290, 301
 - manually perform an animation 172
 - Map interface
 - keySet method **113**
 - Marketwire 307
 - mashup **6**
 - Master/Detail Flow template **42**
 - match_parent value of the Layout height property 103
 - match_parent value of the Layout width property 103
 - Max Length property of an EditText 76
 - Max property of a SeekBar 77
 - media files 171
 - MediaStore class **202**
 - MediaStore.Images.Media class 202
 - insertImage method **202**
 - medium sized font 73
 - Menu class **132**, 150, 217
 - menu folder of an Android project **46**, **134**
 - menu name xxiii
 - MenuInflater class **151**, 217, 267
 - inflate method **151**
 - MenuItem class
 - getItemID method **218**
 - merchant account **300**
 - method **17**
 - method call **17**, 17
 - micro blogging 304, 305
 - mobile advertising 296
 - mobile advertising network 297
 - AdMob 297
 - mobile advertising networks 307
 - AdMob 308
 - Flurry 308
 - InMobi 308
 - Jumptap 308
 - Medialets 308
 - mobile advertising networks (cont.)
 - mMedia 308
 - Nexage 308
 - Smaato 308
 - Tapjoy 308
 - mobile payment provider 299
 - Boku 299
 - PayPal Mobile Libraries 299
 - Samsung In-App Purchase 299
 - Zong 299
 - mobile payment providers 298
 - modal dialog **99**
 - MODE_PRIVATE constant **113**
 - MODE_WORLD_READABLE constant **113**
 - MODE_WORLD_WRITABLE constant **113**
 - monetizing apps 290, 297
 - MotionEvent class **172**, 194, **202**, 226
 - getActionIndex method **226**
 - getActionMasked method **226**
 - getPointerCount method **227**
 - getX method 227
 - getY method 227
 - moveTo method of class Path **226**
 - moveToFirst method of class Cursor **280**
 - MP3 player 5
 - multimedia xvii
 - multiple format specifiers 137
 - MultiSelectListPreference class **133**
 - multitouch 225
 - multitouch screen 5
 - music audio stream 172, 181
- N**
- naming convention
 - GUI components 71
 - near-field communication (NFC) 8
 - nested structure of a layout 73
 - nested Views **251**
 - network access 13
 - New Android Application dialog **38**
 - newsgroups 33
 - Android Developers 33
 - notifyDataSetChanged method **117**
 - notifyDataSetChanged method of class ArrayAdapter **117**
 - NumberFormat class **69**, 79
 - format method **83**
 - numbering format specifiers 137
 - numeric input 68
 - numeric keypad 65
- O**
- obfuscate 292
 - object 16
 - object (or instance) 18
 - object-oriented analysis and design (OOAD) **18**
 - object-oriented language **18**
 - object-oriented programming (OOP) **18**
 - object serialization xix
 - Objective-C command xxiii
 - object-oriented analysis and design (OOAD) 18
 - OEM original equipment manufacturer 4
 - onActivityCreated method of class Fragment 176
 - onAttach method of class Fragment **201**, 233, 263, 270, 275
 - onCreate method of class Activity **67**, 171
 - onCreate method of class Fragment **133**, 165
 - onCreate method of class SQLiteOpenHelper **286**
 - onCreateDialog method of class DialogFragment **164**
 - onCreateOptionsMenu method of class Activity **132**, 150

onCreateOptionsMenu
 method of class Fragment
 217, 277

onCreateView method of class
 Fragment **133**, 155, 176

onDestroy method of class
 Activity 171, **172**

onDestroy method of class
 Fragment **172**, 177

onDetach method of class
 Fragment **201**, 233, 263,
 270, 275

onDowngrade method of class
 SQLiteOpenHelper **287**

onDraw method of class View
 223

OnItemClickListener
 interface 264

onOptionsItemSelected
 method of class Activity
 132, 151

onOptionsItemSelected
 method of class Fragment
 217, 277

onPause method of class
 Activity 171, **172**

onPause method of class
 Fragment **172**, 177, 215

onPostExecute method **266**,
 267, 279, 280

onPostExecute method of
 class AsyncTask **266**, 267,
 279, 280

onProgressUpdate method
 266, 279

onProgressUpdate method of
 class AsyncTask **266**, 279

onResume method of class
 Activity 171

onResume method of class
 Fragment 265, 277

onSaveInstanceState
 method of class Fragment
 246, 277

on-screen component xxiii

OnSeekBarChangeListener
 interface 84

onSensorChanged method
 215

onSensorChanged method of
 interface
 SensorEventListener **215**

onSingleTap method of class
 GestureDetector.Simple
 GestureListener **224**

onSizeChanged method of
 class View **182**, 221

onStart method of class
 Activity **150**, 171

onStart method of class
 Fragment **214**

onStop method of class
 Activity 171

onStop method of class
 Fragment **267**

onTouchEvent method of class
 View 225

onTouchEvent method of class
 View **172**, 194, **202**

onUpgrade method of class
 SQLiteOpenHelper **286**

onCreateView method of
 class Fragment **263**

OOAD (object-oriented
 analysis and design) 18

OOP (object-oriented
 programming) **18**

Open Handset Alliance 7

open source 3

open source apps 4

Open Source Project discussion
 groups 3

opening a database 282

openPR 307

operating system 7

operating system requirements
 xxiii

operating systems services 13

options menu 19, 24, 126, 128,
 200

Orientation property of a
 GridLayout 72

orientation sensor 201

original equipment
 manufacturer (OEM) 4

Outline window 73, 102

Outline window in Eclipse 66,
 68

P

package 12

Package Explorer window
 171, 245

Packages

- android.app 13, **67**, 80,
110, 132, 133
- android.content 13, **98**,
110, 202
- android.content.res 13,
134, **149**, **157**
- android.database 13,
247
- android.database.sqlite
 13, **247**
- android.graphics 13,
173, 202
- android.graphics.
 drawable 13, 162
- android.hardware 13
- android.media 13, 172
- android.net 13, **110**
- android.os 13, 80, 135
- android.preference 13,
132
- android.provider 13
- android.text 13, **69**, 80
- android.util 13, 136, 180
- android.view 13, **111**,
132, 172, 202
- android.view.
 animation 135
- android.view.
 inputmethod **111**
- android.widget **111**,
80, 111, 135
- java.io 13, 162
- java.text 13, **69**, 79
- java.util 13, 136

padding element of a shape
 250

Padding property of a view 77

paid app
 average price 296

Paint class **173**

- filled shape with a border
221
- filled shape without a border
221
- line **221**

Paint class (cont.)
 setAntiAlias method **220**
 setStrokeCap method
 221, **237**
 setStrokeWidth method
 221
 setStyle method **220**
 styles **221**
 parse method of class Uri **119**
 Path class **202**
 moveTo method **226**
 quadTo method **227**
 reset method **226**
 payment 300
 payment processor 296
 persistent key/value pairs 110
 photo sharing 305
 Photo Sphere 10
 piracy 293
 play method of class
 SoundPool **186**
Play Store app 302
 pointer (for touch events) 225
 pop the back stack 258
 popBackStack method of class
 FragmentManager **258**
 portrait mode 182
 portrait orientation 56, 70, 88
 postDelayed method of class
 Handler **135**, 164
 PR Leap 307
 Preference class **133**
 PreferenceFragment class
132, 165
 addPreferencesFromResource method 165
 PreferenceManager class **133**,
 149
 setDefaultValues
 method 149, **149**
Preparing for Release 290
 press release writing 307
 pressure sensor 201
 prevent the soft keyboard from
 being displayed at app startup
 124
 prevent the soft keyboard from
 displaying when app loads
 100
 price 296

pricing your app 295
 printBitmap method of class
 PrintHelper **230**
 PrintHelper class 230
 printBitmap method **230**
 PrintHelper.SCALE_MODE_
 FILL 230
 PrintHelper.SCALE_MODE_
 FIT 230
 private key **293**
 PRLog 307
 programmatically create GUI
 components 133
Progress property of a
 SeekBar 77
 ProGuard **292**
 project **38**
 project templates **42**
 Blank Activity **42**
 Fullscreen Activity **42**
 Master-Detail Application
42
 project, add a class 175
Properties window 49, 50, 51,
 52, 54
 property animation xix, 134,
 146
 proximity sensor 201
 public relations 306
 publish a new version of an app
 303
 publishing data on an Android
 device 13
 push onto the back stack **258**
 putExtra method of class
 Intent **122**
 putLong method of class
 Bundle **258**
 putString method of class
 SharedPreferences. Edit
 or **117**

Q
 quadratic bezier curve 227
 quadTo method of class Path
227
 query method of class
 SQLiteDatabase **284**

R
 R class **82**
 R.drawable class **82**
 R.id class **83**
 R.layout class **83**
 R.layout.activity_main
 constant **83**, 112
 R.string class **83**
 raw folder of an Android project
46, **134**
 recent apps button 24
 red method of class Color **233**
 redraw a View 223
 registerListener method of
 class SensorManager **214**
 registerOnSharedPreference
 ChangeListener method
 of class SharedPreferences
149
 RelativeLayout **46**
 release method of class
 SoundPool **192**
 release resources 280
 remove apps from Market 303
 rendering and tracking text 13
 replace method of class
 FragmentTransaction **258**
 reporting bugs 3
 requirements **18**
 res folder of an Android project
45, **50**
 res/drawable-mdpi folder
 250
 res/raw folder of an Android
 project **171**, 175
 reset method of class Path
226
 resource 301
Resource Chooser dialog 50,
 51, 52
 resources 60
 alternative-resource naming
 conventions 59
 android-developers.
 blogspot.com/ 34
 androiddevweekly.com/
 34

resources (cont.)

answers.oreilly.com/
topic/862-ten-tips-
for-android-
application-
development/ 34
code.google.com/p/
apps-for-android/ 34
cyrilmottier.com/ 34
default 59
developer.motorola.com
34
developer.sprint.com/
site/global/develop/
mobile_platforms/
android/android.jsp
34
graphics-geek.
blogspot.com/ 34
Localization Checklist 62
localized 59
stackoverflow.com/
tags/android/
topusers 34
style **246**
www.brighthub.com/
mobile/google-
android.aspx 34
www.curious-
creature.org/
category/android/ 34
www.htcdev.com/ 34
Resources class **149**, 157
getConfiguration
method **149**
getString method **157**
restrict maximum number of
digits in an EditText 68
returning false from an event
handler **224**
reusable software components
16
Reuse **17**
reuse 17
reverse engineering 292
RGB 25
RGB values **77**
rotate animation for a View
146
rotation vector sensor 201

rule markers (Android
Developer Tools) 54
Runnable interface 135, 192
runOnUiThread method of
class Activity **192**

S
saved state 82
scale animation for a View
146
scale mode 230
Scale Type property of an
ImageView 142
SCALE_MODE_FILL 230
SCALE_MODE_FIT 230
scale-independent pixels 139
scale-independent pixels (sp) **52**
screen capture 293, 294
screenshot specifications 293
scroll touch event 202
scrollable list of items 98, 247
ScrollView class **251**
search operators (Twitter) 90
SeekBar
Max property 77
Progress property 77
SeekBar class 66, **68**, 80
SeekBar.OnSeekBarChangeL
istener interface 69, 80,
234
send a message to an object 17
Sensor class **201**
Sensor Simulator 15
SENSOR_DELAY_NORMAL
constant of class
SensorManager 214
Sensor.TYPE_ACCELEROMETER
constant 214
SensorEvent class **216**
SensorEventListener
interface 215
SensorEventListener
listener **215**
SensorManager class 214
getDefaultSensor
method **214**
registerListener
method **214**
unregisterListener
method **215**

SensorManager.SENSOR_
DELAY_NORMAL constant 214
sensors
accelerometer **201**, 215
gravity 201
gyroscope 201
light 201
linear acceleration 201
magnetic field 201
orientation 201
pressure 201
proximity 201
rotation vector 201
temperature 201
set in an animation **146**
Set interface **136**
setAntiAlias method of class
Paint **220**
setArguments method of class
Fragment **258**
setBackground-color method
234
setBackground-color method
of class View **234**
setChoiceMode method of
class ListView **264**
setContentView method of
class Activity **83**
setDefaultValues method of
class PreferenceManager
149
setEmptyText method of class
ListFragment **264**
setImageBitmap method of
class View **238**
setImageDrawable method of
class InputStream **162**
setListAdapter method of
class ListActivity **114**
setListAdapter method of
class ListFragment **265**
setRepeatCount method of
class Animation **136**, 157
setRequestedOrientation
method of class Activity
149
setRetainInstance method
of class Fragment **264**
setStrokeCap method of class
Paint 221, **237**

- setStrokeWidth method of class Paint 221
- setStyle method of class Paint **220**
- setSystemUiVisibility method of class View **224**
- Setting hardware emulation options** 30
- setVolumeControlStream method of class Activity **172, 176**
- shape element **250**
- SharedPreferences class **98, 110, 111**
 - edit method **117**
 - getAll method **113**
 - getString method **118**
 - getStringSet method **153**
 - registerOnSharedPreferenceChangeListener method **149**
- SharedPreferences.Editor class **98, 117**
 - apply method **117**
 - putString method **117**
- show method of class DialogFragment **164**
- shuffle a collection 162
- shuffle method of class Collections **136**
- signing apps 290
- simple collision detection 186
- simple touch events 172
- SimpleCursorAdapter class **264**
- SimpleOnGestureListener interface 224
- single-screen app 42
- slider 68
- SMS 96
- Social API 9
- social media sites 304
- social networking 304, 305
- soft buttons on an Android device **24**
- soft keyboard
 - prevent display at app startup 124
 - prevent from displaying when app loads 100
- soft keyboard (cont.)
 - remain on screen 70
 - types 253
- soft keypad 88
- sort
 - case insensitive 114
- sort method of class Collections **114**
- sound effects 172
- sound files 175
- sound quality 181
- SoundPool class **172, 181**
 - load method **182**
 - play method **186**
 - release method **192**
- sounds 171
- source code 2
- source-code listing 2
- sp (scale-independent pixels) 52
- speech recognition xix
- speech synthesis xix
- SQL (Structured Query Language) 247
- SQLite 13, 242, 247
- SQLiteDatabase class **247**
 - delete method **285**
 - execSQL method **287**
 - insert method **283**
 - query method **284**
 - update method **284**
- SQLiteOpenHelper class **247, 282, 286**
 - getWritableDatabase method **282**
 - onCreate method **286**
 - onDowngrade method **287**
 - onUpgrade method **286**
- SQLiteOpenHelper class
 - close method **283**
- star ratings for apps 303
- startActivity method of class Context **99, 119**
- startAnimation method of class View **136**
- stream for playing music 181
- streaming 13
- String resource
 - containing multiple format specifiers 137
- String.CASE_INSENSITIVE_ORDER 114
- strings.xml **50, 75, 102**
- stroke element of a shape **250**
- Structured Query Language (SQL) 247
- style attribute of a GUI component **246**
- Style** property of a View 252, 254
- style resource 252, 254
- style resources **246**
- styles.xml 249
- subclass 67
- support both portrait and landscape orientations 103
- surfaceChanged method of interface SurfaceHolder.Callback **193**
- surfaceCreated method of interface SurfaceHolder.Callback **193**
- surfaceDestroyed method of interface SurfaceHolder.Callback **193**
- SurfaceHolder class **173, 181**
 - addCallback method **181**
 - lockCanvas method **196**
- SurfaceHolder.Callback interface **173, 181, 193**
 - surfaceChanged method **193**
 - surfaceCreated method **193**
 - surfaceDestroyed method **193**
- SurfaceView class **173, 181**
 - getHolder method **181**
- synchronized 196
- syntax coloring xvii, 2
- system bar 36, 98, 246
- SYSTEM_UI_FLAG_FULLSCREEN 224
- SYSTEM_UI_FLAG_HIDE_NAVIGATION 224
- SYSTEM_UI_FLAG_IMMERSIVE 224

SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN 224
 SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION 224
 SYSTEM_UI_FLAG_LAYOUT_STABLE 224

T

TableLayout class **70**
 tablet 8
 TalkBack **37**, **57**, 103, 105
 Localization 62
 temperature sensor 201
Text Appearance property of a
 TextView 73
 text box 68
Text Color property of a
 component **53**
 text field 68
Text property of a component
 50
Text Size property of a
 component **52**
 Text-to-Speech API 9
 TextView class **37**, 50, 68, 80
 Label For property 76
 Text Appearance property
 73
 TextView component **47**
 TextWatcher interface **69**, 80
 Theme
 Holo Dark 40
 Holo Light 40
 Holo Light with dark action
 bars theme 40
 thread (for animation) 172
 Thread class 195
 thread safe GUI 135
 Threadr class 172
Tip Calculator app 15
 Toast class **135**, 153
 makeText method **153**
 Tools
 logcat **136**
 touch event 202, **225**
 touch events
 fling 202
 long press 202
 scroll 202
 simple 172

track app installs 303
 translate animation
 android:duration
 attribute **147**
 android:fromXDelta
 attribute **146**
 android:startOffset
 attribute **147**
 android:toXDelta
 attribute **146**
 translate animation for a
 View **146**
 transparency 77, 200
 tweened animation 134, **146**
 tweet 305
 Twitter 6, 96, **305**
 @deitel 305
 hashtag 305
 tweet 305
 Twitter search 90
 operators 92
 TYPE_ACCELEROMETER
 constant of class Sensor 214

U

unregisterListener method
 of class SensorManager **215**
 update method of class
 SQLiteDatabase **284**
 upgrading a database 282
 Uri class **110**, **119**
 parse method **119**
 URL encoded String 118
USB debugging 30
Use Default Margins property
 of a GridLayout 73
 utilities 31

V

values folder of an Android
 project **46**, **50**
 version code 292
 version name 292
 versioning your app 290
Versioning Your Applications 292
 video xix, 13
 video sharing 305
 view 67
 View animations **146**

View class **111**, 173, **234**
 custom subclass 178
 getSystemUiVisibility
 method **224**
 invalidate method **222**
 onDraw method **223**
 onSizeChanged method
 182, 221
 onTouchEvent method
 172, 194, **202**, 225
 redraw a View 223
 setImageBitmap method
 238
 setSystemUiVisibility
 method **224**
 size changes 182
 startAnimation method
 136
 View.OnClickListener
 interface **111**
 View.SYSTEM_UI_FLAG_FULLSCREEN 224
 View.SYSTEM_UI_FLAG_HIDE_NAVIGATION 224
 View.SYSTEM_UI_FLAG_IMMERSIVE 224
 View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN 224
 View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
 224
 View.SYSTEM_UI_FLAG_LAYOUT_STABLE 224
 ViewGroup class **251**
 viral marketing 304, 305
 viral video 305
 virtual camera operator **9**
 virtual goods 298
 VoiceOver
 enable/disable 58
 volume 172

W

web services **6**
 Amazon eCommerce 6
 eBay 6
 Facebook 6
 Flickr 6
 Foursquare 6
 Google Maps 6

web services (cont).
Groupon 6
Instagram 6
Last.fm 6
LinkedIn 6
Microsoft Bing 6
Netflix 6
PayPal 6
Salesforce.com 6
Skype 6
Twitter 6
WeatherBug 6
Wikipedia 6
Yahoo Search 6
YouTube 6
Zillow 6
Weight property of a
component 78, 105

Weight property of a GUI
component 141
Welcome app 14, 15
Welcome tab in Eclipse 38
widget 13, 80, 111
width of a column 70
Wi-Fi Direct 9
Window soft input mode
option 88, 124
WindowManager class **135**, 150
Windows 14
workspace **19**
Workspace Launcher window
19
wrap_content value of the
`android:layout_height`
attribute 76, 77

wrap_content value of the
`android:layout_width`
attribute 76, 77
`www.deitel.com/training`
309

X

xml folder of an Android project
46, 134
XML utilities 13

Y

YouTube 294