

2ND EDITION

Android Programming

THE BIG NERD RANCH GUIDE

Bill Phillips, Chris Stewart, Brian Hardy
& Kristin Marsicano

Android Programming: The Big Nerd Ranch Guide

by Bill Phillips, Chris Stewart, Brian Hardy and Kristin Marsicano

Copyright © 2015 Big Nerd Ranch, LLC.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, contact

Big Nerd Ranch, LLC.
200 Arizona Ave NE
Atlanta, GA 30307
(770) 817-6373
<http://www.bignerdranch.com/>
book-comments@bignerdranch.com

The 10-gallon hat with propeller logo is a trademark of Big Nerd Ranch, Inc.

Exclusive worldwide distribution of the English edition of this book by

Pearson Technology Group
800 East 96th Street
Indianapolis, IN 46240 USA
<http://www.informit.com>

The authors and publisher have taken care in writing and printing 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.

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.

ISBN-10 0134171497
ISBN-13 978-0134171494

Second edition, first printing, August 2015
Release D.2.1.1

Dedication

To God, or to whatever it is that you personally have faith in. Reader, I hope that you find the many explanations in this book useful. Please don't ask me how they got here, though. I once thought that I was responsible. Fortunately for you, I was wrong.

— B.P.

To my dad, David, for teaching me the value of hard work. To my mom, Lisa, for pushing me to always do the right thing.

— C.S.

For Donovan. May he live a life filled with activities and know when to use fragments.

— B.H.

*To my dad, Dave Vadas, for inspiring and encouraging me to pursue a career in computing. And to my mom, Joan Vadas, for cheering me on through all the ups and downs (and for reminding me that watching an episode of *The Golden Girls* always makes things better).*

— K.M.

This page intentionally left blank

Acknowledgments

We feel a bit sheepish having our names on the cover of this book. The truth is that without an army of collaborators, this book could never have happened. We owe them all a debt of gratitude.

- Our co-instructors and members of our Android development team, Andrew Lunsford, Bolot Kerimbaev, Brian Gardner, David Greenhalgh, Jason Atwood, Josh Skeen, Kurt Nelson, Matt Compton, Paul Turner, and Sean Farrell. We thank them for their patience in teaching work-in-progress material, as well as their suggestions and corrections. If we could give ourselves additional brains to do with as we pleased, we would not. We would just put the new brains in a big pile, and share them with our colleagues. We trust them at least as much as we trust our own selves.
- Special thanks to Sean Farrell for graciously updating many screen shots as Android Studio evolved, and to Matt Compton for publishing all of our sample apps to the Google Play Store.
- Kar Loong Wong and Zack Simon, members of Big Nerd Ranch's amazing design team. Kar made BeatBox look intimidating and polished, and provided advice and imagery for the material design chapter. Zack took time out of his schedule to design MockWalker for us. Kar and Zack's design abilities seem like unknowable superpowers to us. We thank them, and bid them fond returns to their home planet.
- Our technical reviewers, Frank Robles and Roy Kravitz, who helped us find and fix flaws.
- Thanks to Aaron Hillegass. Aaron's faith in people is one of the great and terrifying forces of nature. Without it, we would never have had the opportunity to write this book, nor would we ever have completed it. (He also gave us money, which was very friendly of him.)
- Our editor, Elizabeth Holaday, who many times saved us from going down rabbit holes. She kept our writing focused on what our readers actually care about and spared you all from confusing, boring, and irrelevant detours. Thank you, Liz, for being organized and patient, and for being a constant supportive presence, even though you live many miles away.
- Ellie Volckhausen, who designed our cover.
- Simone Payment, our copy-editor, who found and smoothed rough spots.
- Chris Loper at IntelligentEnglish.com, who designed and produced the print book and the EPUB and Kindle versions. His DocBook toolchain made life much easier, too.

Finally, thanks to our students. We wish that we had room to thank every single student who gave us a correction or opinion on the book as it was shaping up. It is your curiosity we have worked to satisfy, your confusions we have worked to clarify. Thank you.

This page intentionally left blank

Table of Contents

Learning Android	xvii
Prerequisites	xvii
What's New in the Second Edition?	xvii
How to Use This Book	xviii
How This Book is Organized	xviii
Challenges	xix
Are you more curious?	xix
Code Style	xix
Typographical Conventions	xx
Android Versions	xx
The Necessary Tools	xxi
Downloading and Installing Android Studio	xxi
Downloading Earlier SDK Versions	xxi
An Alternative Emulator	xxii
A Hardware Device	xxii
1. Your First Android Application	1
App Basics	2
Creating an Android Project	2
Navigating in Android Studio	8
Laying Out the User Interface	9
The view hierarchy	13
Widget attributes	14
Creating string resources	15
Previewing the layout	15
From Layout XML to View Objects	16
Resources and resource IDs	18
Wiring Up Widgets	20
Getting references to widgets	21
Setting listeners	22
Making Toasts	23
Using code completion	25
Running on the Emulator	26
For the More Curious: Android Build Process	29
Android build tools	31
2. Android and Model-View-Controller	33
Creating a New Class	34
Generating getters and setters	34
Model-View-Controller and Android	37
Benefits of MVC	38
Updating the View Layer	39
Updating the Controller Layer	41
Running on a Device	46
Connecting your device	46
Configuring your device for development	47
Adding an Icon	48

- Adding resources to a project 49
- Referencing resources in XML 52
- Challenges 53
- Challenge: Add a Listener to the TextView 53
- Challenge: Add a Previous Button 54
- Challenge: From Button to ImageButton 55
- 3. The Activity Lifecycle 57
 - Logging the Activity Lifecycle 58
 - Making log messages 58
 - Using LogCat 60
 - Rotation and the Activity Lifecycle 63
 - Device configurations and alternative resources 64
 - Saving Data Across Rotation 68
 - Overriding onSaveInstanceState(Bundle) 69
 - The Activity Lifecycle, Revisited 70
 - For the More Curious: Testing onSaveInstanceState(Bundle) 72
 - For the More Curious: Logging Levels and Methods 73
- 4. Debugging Android Apps 75
 - Exceptions and Stack Traces 76
 - Diagnosing misbehaviors 77
 - Logging stack traces 78
 - Setting breakpoints 79
 - Using exception breakpoints 82
 - Android-Specific Debugging 84
 - Using Android Lint 84
 - Issues with the R class 85
- 5. Your Second Activity 87
 - Setting Up a Second Activity 88
 - Creating a new activity 89
 - A new activity subclass 92
 - Declaring activities in the manifest 92
 - Adding a Cheat! button to QuizActivity 93
 - Starting an Activity 95
 - Communicating with intents 96
 - Passing Data Between Activities 97
 - Using intent extras 98
 - Getting a result back from a child activity 101
 - How Android Sees Your Activities 106
 - Challenge 109
- 6. Android SDK Versions and Compatibility 111
 - Android SDK Versions 111
 - Compatibility and Android Programming 112
 - A sane minimum 112
 - Minimum SDK version 114
 - Target SDK version 114
 - Compile SDK version 114
 - Adding code from later APIs safely 114
 - Using the Android Developer Documentation 117

Challenge: Reporting the Build Version	119
7. UI Fragments and the Fragment Manager	121
The Need for UI Flexibility	122
Introducing Fragments	123
Starting CriminalIntent	124
Creating a new project	126
Fragments and the support library	128
Adding dependencies in Android Studio	129
Creating the Crime class	132
Hosting a UI Fragment	133
The fragment lifecycle	133
Two approaches to hosting	134
Defining a container view	135
Creating a UI Fragment	136
Defining CrimeFragment's layout	136
Creating the CrimeFragment class	138
Adding a UI Fragment to the FragmentManager	142
Fragment transactions	143
The FragmentManager and the fragment lifecycle	145
Application Architecture with Fragments	146
The reason all our activities will use fragments	147
For the More Curious: Why Support Fragments are Superior	148
For the More Curious: Using Built-In Fragments	148
8. Creating User Interfaces with Layouts and Widgets	149
Upgrading Crime	149
Updating the Layout	150
Wiring Widgets	153
More on XML Layout Attributes	154
Styles, themes, and theme attributes	154
Screen pixel densities and dp and sp	155
Android's design guidelines	156
Layout parameters	157
Margins vs. padding	157
Using the Graphical Layout Tool	158
Creating a landscape layout	160
Adding a new widget	161
Editing attributes in properties view	161
Reorganizing widgets in the component tree	162
Updating child layout parameters	163
How android:layout_weight works	164
The graphical layout tool and you	165
Widget IDs and multiple layouts	166
Challenge: Formatting the Date	166
9. Displaying Lists with RecyclerView	167
Updating CriminalIntent's Model Layer	168
Singletons and centralized data storage	168
An Abstract Activity for Hosting a Fragment	171
A generic fragment-hosting layout	171

- An abstract Activity class 172
- RecyclerView, Adapter, and ViewHolder 176
 - ViewHolders and Adapters 177
 - Using a RecyclerView 180
 - Implementing an Adapter and ViewHolder 182
- Customizing List Items 185
 - Creating the list item layout 185
 - Using a custom item view 188
- Responding to Presses 190
 - For the More Curious: ListView and GridView 191
 - For the More Curious: Singletons 192
- 10. Using Fragment Arguments 193
 - Starting an Activity from a Fragment 193
 - Putting an extra 194
 - Retrieving an extra 195
 - Updating CrimeFragment’s view with Crime data 196
 - The downside to direct retrieval 197
 - Fragment Arguments 197
 - Attaching arguments to a fragment 198
 - Retrieving arguments 199
 - Reloading the List 200
 - Getting Results with Fragments 202
 - Challenge: Efficient RecyclerView Reloading 203
 - For the More Curious: Why Use Fragment Arguments? 204
- 11. Using ViewPager 205
 - Creating CrimePagerActivity 206
 - ViewPager and PagerAdapter 207
 - Integrating CrimePagerActivity 208
 - FragmentManagerAdapter vs. FragmentPagerAdapter 211
 - For the More Curious: How ViewPager Really Works 212
 - For the More Curious: Laying Out Views in Code 213
- 12. Dialogs 215
 - The AppCompatActivity Library 216
 - Creating a DialogFragment 217
 - Showing a DialogFragment 220
 - Setting a dialog’s contents 221
 - Passing Data Between Two Fragments 224
 - Passing data to DatePickerFragment 225
 - Returning data to CrimeFragment 226
 - Challenge: More Dialogs 233
 - Challenge: A Responsive DialogFragment 233
- 13. The Toolbar 235
 - AppCompatActivity 235
 - Using the AppCompatActivity library 236
 - Menus 238
 - Defining a menu in XML 239
 - Creating the menu 244
 - Responding to menu selections 246

Enabling Hierarchical Navigation	248
How hierarchical navigation works	249
An Alternative Action Item	249
Toggling the action item title	251
“Just one more thing...”	252
For the More Curious: Toolbar vs Action Bar	254
Challenge: Deleting Crimes	255
Challenge: Plural String Resources	255
Challenge: An Empty View for the RecyclerView	255
14. SQLite Databases	257
Defining a Schema	257
Building Your Initial Database	258
Debugging database issues	261
Gutting CrimeLab	262
Writing to the Database	263
Using ContentValues	263
Inserting and updating rows	264
Reading from the Database	266
Using a CursorWrapper	267
Converting to model objects	269
For the More Curious: More Databases	271
For the More Curious: The Application Context	272
Challenge: Deleting Crimes	272
15. Implicit Intents	273
Adding Buttons	274
Adding a Suspect to the Model Layer	276
Using a Format String	278
Using Implicit Intents	279
Parts of an implicit intent	280
Sending a crime report	281
Asking Android for a contact	283
Checking for responding activities	287
Challenge: ShareCompat	289
Challenge: Another Implicit Intent	289
16. Taking Pictures with Intents	291
A Place for Your Photo	291
Including layout files	292
External Storage	294
Designating a picture location	296
Using a Camera Intent	297
External storage permission	298
Firing the intent	299
Scaling and Displaying Bitmaps	301
Declaring Features	304
For the More Curious: Using Includes	304
Challenge: Detail Display	305
Challenge: Efficient Thumbnail Load	305
17. Two-Pane Master-Detail Interfaces	307

- Adding Layout Flexibility 308
 - Modifying SingleFragmentActivity 309
 - Creating a layout with two fragment containers 309
 - Using an alias resource 311
 - Creating tablet alternatives 312
- Activity: Fragment Boss 314
 - Fragment callback interfaces 314
- For the More Curious: More on Determining Device Size 323
- 18. Assets 325
 - Why Assets, Not Resources 326
 - Creating BeatBox 326
 - Importing Assets 329
 - Getting at Assets 331
 - Wiring Up Assets for Use 333
 - Accessing Assets 336
 - For the More Curious: Non-Assets? 337
- 19. Audio Playback with SoundPool 339
 - Creating a SoundPool 339
 - Loading Sounds 340
 - Playing Sounds 341
 - Unloading Sounds 343
 - Rotation and Object Continuity 344
 - Retaining a fragment 345
 - Rotation and retained fragments 346
 - For the More Curious: Whether to Retain 348
 - For the More Curious: More on Rotation Handling 349
- 20. Styles and Themes 353
 - Color Resources 353
 - Styles 354
 - Style inheritance 355
 - Themes 357
 - Modifying the theme 357
 - Adding Theme Colors 359
 - Overriding Theme Attributes 360
 - Theme spelunking 361
 - Modifying Button Attributes 365
 - For the More Curious: More on Style Inheritance 367
 - For the More Curious: Accessing Theme Attributes 368
 - Challenge: An Appropriate Base Theme 368
- 21. XML Drawables 369
 - Making Uniform Buttons 369
 - Shape Drawables 371
 - State List Drawables 372
 - Layer List Drawables 374
 - For the More Curious: Why Bother with XML Drawables? 376
 - For the More Curious: 9-Patch Images 376
 - For the More Curious: Mipmap Images 381
- 22. More About Intents and Tasks 383

Setting Up NerdLauncher	384
Resolving an Implicit Intent	386
Creating Explicit Intents at Runtime	391
Tasks and the Back Stack	393
Switching between tasks	393
Starting a new task	395
Using NerdLauncher as a Home Screen	397
Challenge: Icons	398
For the More Curious: Processes vs. Tasks	398
For the More Curious: Concurrent Documents	401
23. HTTP & Background Tasks	405
Creating PhotoGallery	406
Networking Basics	409
Asking permission to network	411
Using AsyncTask to Run on a Background Thread	411
You and Your Main Thread	413
Beyond the main thread	414
Fetching JSON from Flickr	415
Parsing JSON text	419
From AsyncTask Back to the Main Thread	422
Cleaning Up AsyncTasks	425
For the More Curious: More on AsyncTask	426
For the More Curious: Alternatives to AsyncTask	427
Challenge: Gson	428
Challenge: Paging	428
Challenge: Dynamically Adjusting the Number of Columns	428
24. Loopers, Handlers, and HandlerThread	429
Preparing RecyclerView to Display Images	429
Downloading Lots of Small Things	432
Communicating with the Main Thread	432
Assembling a Background Thread	433
Messages and Message Handlers	435
Message anatomy	435
Handler anatomy	436
Using handlers	437
Passing handlers	441
For the More Curious: AsyncTask vs. Threads	447
Challenge: Preloading and Caching	447
For the More Curious: Solving the Image Downloading Problem	448
25. Search	449
Searching Flickr	449
Using SearchView	455
Responding to SearchView user interactions	458
Simple Persistence with Shared Preferences	460
Polishing Your App	464
Challenge: Polishing Your App Some More	465
26. Background Services	467
Creating an IntentService	467

- What Services are For 469
 - Safe background networking 470
- Looking for New Results 471
- Delayed Execution with AlarmManager 473
 - Being a good citizen: using alarms the right way 475
 - PendingIntent 477
 - Managing alarms with PendingIntent 477
- Controlling Your Alarm 478
- Notifications 481
- Challenge: Notifications on Android Wear 483
- For the More Curious: Service Details 483
 - What a service does (and does not) do 483
 - A service’s lifecycle 484
 - Non-sticky services 484
 - Sticky services 484
 - Bound services 485
- For the More Curious: JobScheduler and JobServices 486
- For the More Curious: Sync Adapters 488
- Challenge: Using JobService on Lollipop 490
- 27. Broadcast Intents 491
 - Regular Intents vs. Broadcast Intents 491
 - Receiving a System Broadcast: Waking Up on Boot 492
 - Creating and registering a standalone receiver 492
 - Using receivers 495
 - Filtering Foreground Notifications 496
 - Sending broadcast intents 497
 - Creating and registering a dynamic receiver 497
 - Limiting broadcasts to your app using private permissions 500
 - Passing and receiving data with ordered broadcasts 502
 - Receivers and Long-Running Tasks 507
 - For the More Curious: Local Events 507
 - Using EventBus 507
 - Using RxJava 508
 - For the More Curious: Detecting the Visibility of Your Fragment 509
- 28. Browsing the Web and WebView 511
 - One Last Bit of Flickr Data 511
 - The Easy Way: Implicit Intents 514
 - The Harder Way: WebView 516
 - Using WebChromeClient to spruce things up 520
 - Proper Rotation with WebView 522
 - Dangers of handling configuration changes 523
 - For the More Curious: Injecting JavaScript Objects 523
 - For the More Curious: KitKat’s WebView Overhaul 524
 - Challenge: Using the Back Button for Browser History 524
 - Challenge: Supporting Non-HTTP Links 525
- 29. Custom Views and Touch Events 527
 - Setting Up the DragAndDraw Project 527
 - Setting up DragAndDrawActivity 528

Setting up DragAndDrawFragment	528
Creating a Custom View	530
Creating BoxDrawingView	530
Handling Touch Events	532
Tracking across motion events	534
Rendering Inside onDraw(...)	536
Challenge: Saving State	538
Challenge: Rotating Boxes	538
30. Property Animation	539
Building the Scene	539
Simple Property Animation	542
View transformation properties	544
Using different interpolators	546
Color evaluation	546
Playing Animators Together	548
For the More Curious: Other Animation APIs	550
Legacy animation tools	550
Transitions	550
Challenges	550
31. Locations and Play Services	551
Locations and Libraries	551
Google Play Services	552
Creating Locatr	552
Play Services and Location Testing on Emulators	553
Mock location data	554
Building out Locatr	556
Setting Up Google Play Services	559
Location permissions	560
Using Google Play Services	561
Flickr Geosearch	563
Getting a Location Fix	564
Find and Display an Image	566
Challenge: Progress	569
32. Maps	571
Importing Play Services Maps	571
Mapping on Android	571
Maps API Setup	572
Getting a Maps API Key	572
Setting Up Your Map	574
Getting More Location Data	576
Working with Your Map	579
Drawing on the map	582
For the More Curious: Teams and API Keys	584
33. Material Design	587
Material Surfaces	587
Elevation and Z values	589
State list animators	590
Animation Tools	591

- Circular reveal 591
- Shared element transitions 593
- View Components 597
 - Cards 597
 - Floating action buttons 598
 - Snackbars 600
- More on Material Design 601
- 34. Afterword 603
 - The Final Challenge 603
 - Shameless Plugs 603
 - Thank You 604
- Index 605

Learning Android

As a beginning Android programmer, you face a steep learning curve. Learning Android is like moving to a foreign city. Even if you speak the language, it will not feel like home at first. Everyone around you seems to understand things that you are missing. Things you already knew turn out to be dead wrong in this new context.

Android has a culture. That culture speaks Java, but knowing Java is not enough. Getting your head around Android requires learning many new ideas and techniques. It helps to have a guide through unfamiliar territory.

That's where we come in. At Big Nerd Ranch, we believe that to be an Android programmer, you must:

- *write* Android applications
- *understand* what you are writing

This guide will help you do both. We have trained hundreds of professional Android programmers using it. We lead you through writing several Android applications, introducing concepts and techniques as needed. When there are rough spots, when some things are tricky or obscure, you will face them head on, and we will do our best to explain why things are the way they are.

This approach allows you to put what you have learned into practice in a working app right away rather than learning a lot of theory and then having to figure out how to apply it all later. You will come away with the experience and understanding you need to get going as an Android developer.

Prerequisites

To use this book, you need to be familiar with Java, including classes and objects, interfaces, listeners, packages, inner classes, anonymous inner classes, and generic classes.

If these ideas do not ring a bell, you will be in the weeds by page 2. Start instead with an introductory Java book and return to this book afterward. There are many excellent introductory books available, so you can choose one based on your programming experience and learning style.

If you are comfortable with object-oriented programming concepts, but your Java is a little rusty, you will probably be OK. We will provide some brief reminders about Java specifics (like interfaces and anonymous inner classes). Keep a Java reference handy in case you need more support as you go through the book.

What's New in the Second Edition?

This second edition shows how to use the Android Studio integrated development environment to write practical applications for Android 5.1 (Lollipop) that are backwards-compatible through Android 4.1 (Jelly Bean). It includes updated coverage of the fundamentals of Android programming as well as new Lollipop tools like the toolbar and material design. It also covers new tools from the support libraries, like **RecyclerView** and Google Play Services, plus some key standard library tools, like **SoundPool**, animations, and assets.

How to Use This Book

This book is not a reference book. Its goal is to get you over the initial hump to where you can get the most out of the reference and recipe books available. It is based on our five-day class at Big Nerd Ranch. As such, it is meant to be worked through from the beginning. Chapters build on each other and skipping around is unproductive.

In our classes, students work through these materials, but they also benefit from the right environment – a dedicated classroom, good food and comfortable board, a group of motivated peers, and an instructor to answer questions.

As a reader, you want your environment to be similar. That means getting a good night’s rest and finding a quiet place to work. These things can help, too:

- Start a reading group with your friends or coworkers.
- Arrange to have blocks of focused time to work on chapters.
- Participate in the forum for this book at <http://forums.bignerdranch.com>.
- Find someone who knows Android to help you out.

How This Book is Organized

As you work through this book, you will write eight Android apps. A couple are very simple and take only a chapter to create. Others are more complex. The longest app spans 11 chapters. All are designed to teach you important concepts and techniques and give you direct experience using them.

<i>GeoQuiz</i>	In your first app, you will explore the fundamentals of Android projects, activities, layouts, and explicit intents.
<i>CriminalIntent</i>	The largest app in the book, CriminalIntent lets you keep a record of your colleagues’ lapses around the office. You will learn to use fragments, master-detail interfaces, list-backed interfaces, menus, the camera, implicit intents, and more.
<i>BeatBox</i>	Intimidate your foes with this app while you learn more about fragments, media playback, themes, and drawables.
<i>NerdLauncher</i>	Building this custom launcher will give you insight into the intent system and tasks.
<i>PhotoGallery</i>	A Flickr client that downloads and displays photos from Flickr’s public feed, this app will take you through services, multithreading, accessing web services, and more.

<i>DragAndDraw</i>	In this simple drawing app, you will learn about handling touch events and creating custom views.
<i>Sunset</i>	In this toy app, you will create a beautiful representation of a sunset over open water while learning about animations.
<i>Locatr</i>	This app lets you query Flickr for pictures around your current location and display them on a map. In it, you will learn how to use location services and maps.

Challenges

Most chapters have a section at the end with exercises for you to work through. This is your opportunity to use what you have learned, explore the documentation, and do some problem solving on your own.

We strongly recommend that you do the challenges. Going off the beaten path and finding your way will solidify your learning and give you confidence with your own projects.

If you get lost, you can always visit <http://forums.bignerdranch.com> for some assistance.

Are you more curious?

There are also sections at the ends of chapters labeled “For the More Curious.” These sections offer deeper explanations or additional information about topics presented in the chapter. The information in these sections is not absolutely essential, but we hope you will find it interesting and useful.

Code Style

There are two areas where our choices differ from what you might see elsewhere in the Android community:

We use anonymous inner classes for listeners.

This is mostly a matter of opinion. We find it makes for cleaner code in the applications in this book because it puts the listener’s method implementations right where you want to see them. In high-performance contexts or large applications, anonymous inner classes may cause problems, but for most circumstances they work fine.

After we introduce fragments in Chapter 7, we use them for all user interfaces.

Fragments are not an absolutely necessary tool but we find that, when used correctly, they are a valuable tool in any Android developer’s toolkit. Once you get comfortable with fragments, they are not that difficult to work with. Fragments have clear advantages over activities that make them worth the effort, including flexibility in building and presenting your user interfaces.

Typographical Conventions

To make this book easier to read, certain items appear in certain fonts. Variables, constants, and types appear in a fixed-width font. Class names, interface names, and method names appear in a bold, fixed-width font.

All code and XML listings are in a fixed-width font. Code or XML that you need to type in is always bold. Code or XML that should be deleted is struck through. For example, in the following method implementation, you are deleting the call to **makeText(...)** and adding the call to **checkAnswer(true)**.

```
@Override
public void onClick(View v) {
    Toast.makeText(QuizActivity.this, R.string.incorrect_toast,
    Toast.LENGTH_SHORT).show();
    checkAnswer(true);
}
```

Android Versions

This book teaches Android development for all widely used versions of Android. As of this writing, that is Android 4.1 (Jelly Bean) - Android 5.1 (Lollipop). While there is a small amount of market-share on older versions of Android, we find that for most developers the amount of effort required to support those versions is not worth the reward. For more info on the support of versions of Android earlier than 4.1 (in particular, Android 2.2 and Android 2.3), see the first edition of this book.

As Android releases new versions, the techniques you learn in this book will continue to work thanks to Android's backwards compatibility support (see Chapter 6 for details). We will keep track of changes at <http://forums.bignerdranch.com> and offer notes on using this book with the latest version.

The Necessary Tools

To get started with this book, you will need Android Studio. Android Studio is an integrated development environment used for Android development that is based off of the popular IntelliJ IDEA.

An install of Android Studio includes:

Android SDK

the latest version of the Android SDK

Android SDK tools and platform-tools

tools for debugging and testing your apps

A system image for the Android emulator

lets you create and test your apps on different virtual devices

As of this writing, Android Studio is under active development and is frequently updated. Be aware that you may find differences between your version of Android Studio and what you see in this book. Visit <http://forums.bignerdranch.com> for help with these differences.

Downloading and Installing Android Studio

Android Studio is available from Android's developer site at <https://developer.android.com/sdk/>.

If you do not already have it installed, you will need to install the Java Development Kit (JDK7), which you can download from <http://www.oracle.com>.

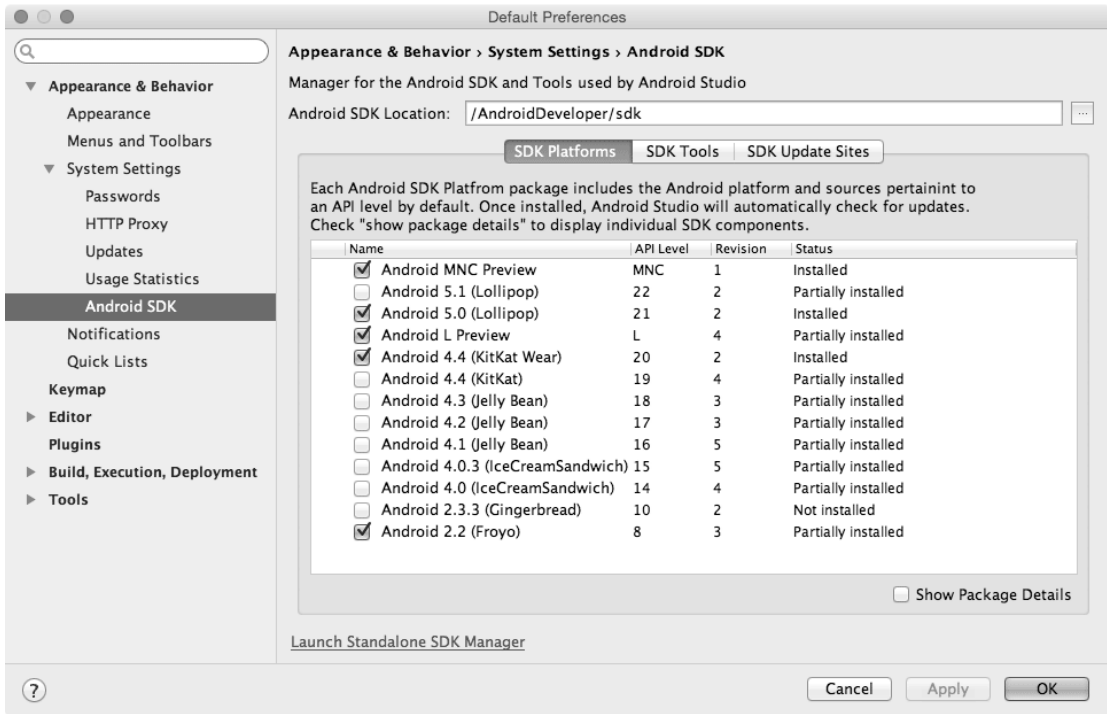
If you are still having problems, return to <https://developer.android.com/sdk/> for more information.

Downloading Earlier SDK Versions

Android Studio provides the SDK and the emulator system image from the latest platform. However, you may want to test your apps on earlier versions of Android.

You can get components for each platform using the Android SDK Manager. In Android Studio, select Tools → Android → SDK Manager. (You will only see the Tools menu if you have a project open. If you have not created a project yet, you can instead access the SDK Manager from the Android Setup Wizard screen. Under the Quick Start section, select Configure → SDK Manager, as shown in Figure 1.)

Figure 1 Android SDK Manager



Select and install each version of Android that you want to test with. Note that downloading these components may take a while.

The Android SDK Manager is also how to get Android’s latest releases, like a new platform or an update of the tools.

An Alternative Emulator

The speed of the Android emulator has improved significantly over time and it is a reasonable way to run the code that you write in this book.

As an alternative, Genymotion is a popular, third-party Android emulator. You will occasionally see references to the Genymotion emulator in this book. For more information on Genymotion, visit <http://genymotion.com/>.

A Hardware Device

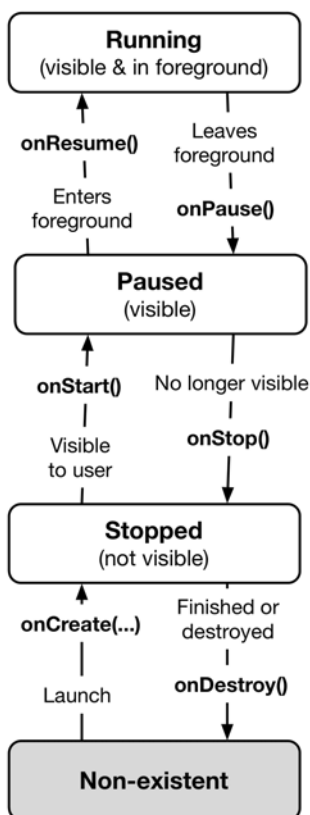
The emulator and Genymotion are useful for testing apps. However, they are no substitute for an actual Android device when measuring performance. If you have a hardware device, we recommend using that device at times when working through this book.

3

The Activity Lifecycle

Every instance of **Activity** has a lifecycle. During this lifecycle, an activity transitions between three states: running, paused, and stopped. For each transition, there is an **Activity** method that notifies the activity of the change in its state. Figure 3.1 shows the activity lifecycle, states, and methods.

Figure 3.1 Activity state diagram



Subclasses of **Activity** can take advantage of the methods named in Figure 3.1 to get work done at critical transitions in the activity's lifecycle.

You are already acquainted with one of these methods – `onCreate(Bundle)`. The OS calls this method after the activity instance is created but before it is put on screen.

Typically, an activity overrides `onCreate(...)` to prepare the specifics of its user interface:

- inflating widgets and putting them on screen (in the call to `setContentView(int)`)
- getting references to inflated widgets
- setting listeners on widgets to handle user interaction
- connecting to external model data

It is important to understand that you never call `onCreate(...)` or any of the other **Activity** lifecycle methods yourself. You override them in your activity subclasses, and Android calls them at the appropriate time.

Logging the Activity Lifecycle

In this section, you are going to override lifecycle methods to eavesdrop on **QuizActivity**'s lifecycle. Each implementation will simply log a message informing you that the method has been called.

Making log messages

In Android, the `android.util.Log` class sends log messages to a shared system-level log. **Log** has several methods for logging messages. Here is the one that you will use most often in this book:

```
public static int d(String tag, String msg)
```

The `d` stands for “debug” and refers to the level of the log message. (There is more about the **Log** levels in the final section of this chapter.) The first parameter identifies the source of the message, and the second is the contents of the message.

The first string is typically a TAG constant with the class name as its value. This makes it easy to determine the source of a particular message.

In `QuizActivity.java`, add a TAG constant to **QuizActivity**:

Listing 3.1 Adding TAG constant (`QuizActivity.java`)

```
public class QuizActivity extends AppCompatActivity {  
    private static final String TAG = "QuizActivity";  
    ...  
}
```

Next, in `onCreate(...)`, call `Log.d(...)` to log a message.

Listing 3.2 Adding log statement to **onCreate(...)** (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);
    }

    ...
}
```

Now override five more methods in **QuizActivity** by adding the following after **onCreate(Bundle)** and before **onCreateOptionsMenu(Menu)**:

Listing 3.3 Overriding more lifecycle methods (QuizActivity.java)

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

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

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

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

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

...
}
```

Notice that you call the superclass implementations before you log your messages. These superclass calls are required. Calling the superclass implementation before you do anything else is critical in **onCreate(...)**; the order is less important in the other methods.

You may have been wondering about the `@Override` annotation. This asks the compiler to ensure that the class actually has the method that you are attempting to override. For example, the compiler would be able to alert you to the following misspelled method name:

```
public class QuizActivity extends AppCompatActivity {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_quiz);  
    }  
  
    ...  
}
```

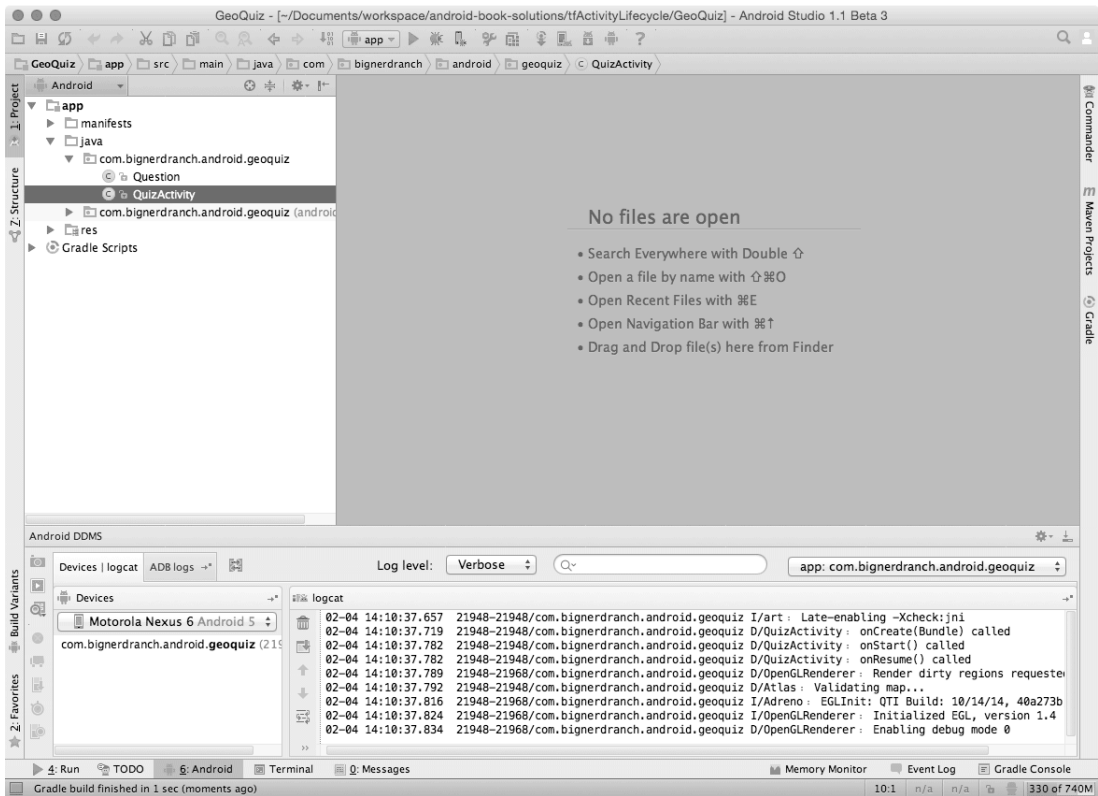
The **Activity** class does not have an **onCreat(Bundle)** method, so the compiler will complain. Then you can fix the typo rather than accidentally implementing **QuizActivity.onCreat(Bundle)**.

Using LogCat

To access the log while the application is running, you can use LogCat, a log viewer included in the Android SDK tools.

When you run GeoQuiz, you should see LogCat appear at the bottom of Android Studio, as shown in Figure 3.2. If LogCat is not visible, select the Android tool window near the bottom of the screen and ensure that the Devices | logcat tab is selected.

Figure 3.2 Android Studio with LogCat

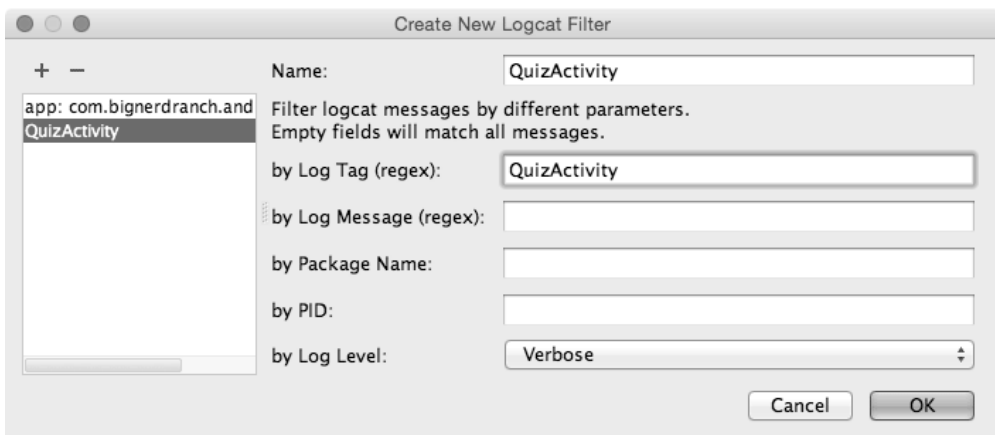


Run GeoQuiz and messages will start materializing in LogCat. By default, log statements that are generated with your app's package name are shown. You will see your own messages along with some system output.

To make your messages easier to find, you can filter the output using the TAG constant. In LogCat, click the filter drop-down box in the top right of the LogCat pane. Notice the existing filter, which is set up to show messages from only your app. Selecting No Filters will show log messages generated from all over the system.

In the filter dropdown, select Edit Filter Configuration. Use the + button to create a brand-new filter. Name the filter **QuizActivity** and enter **QuizActivity** in the by Log Tag: field (Figure 3.3).

Figure 3.3 Creating a filter in LogCat



Click OK, and only messages tagged **QuizActivity** will be visible (Figure 3.4).

Three lifecycle methods were called after GeoQuiz was launched and the initial instance of **QuizActivity** was created.

Figure 3.4 Launching GeoQuiz creates, starts, and resumes an activity

```
logcat
08-04 17:51:02.316 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 17:51:02.347 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 17:51:02.347 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
```

(If you are not seeing the filtered list, select the QuizActivity filter from LogCat's filter dropdown.)

Now let's have some fun. Press the Back button on the device and then check LogCat. Your activity received calls to **onPause()**, **onStop()**, and **onDestroy()** (Figure 3.5).

Figure 3.5 Pressing the Back button destroys the activity

```
logcat
08-04 17:51:02.316 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 17:51:02.347 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 17:51:02.347 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
08-04 17:54:35.463 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onPause() called
08-04 17:54:35.811 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onStop() called
08-04 17:54:35.811 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onDestroy() called
```

When you pressed the Back button, you told Android, “I’m done with this activity, and I won’t need it anymore.” Android then destroyed your activity. This is Android’s way of being frugal with your device’s limited resources.

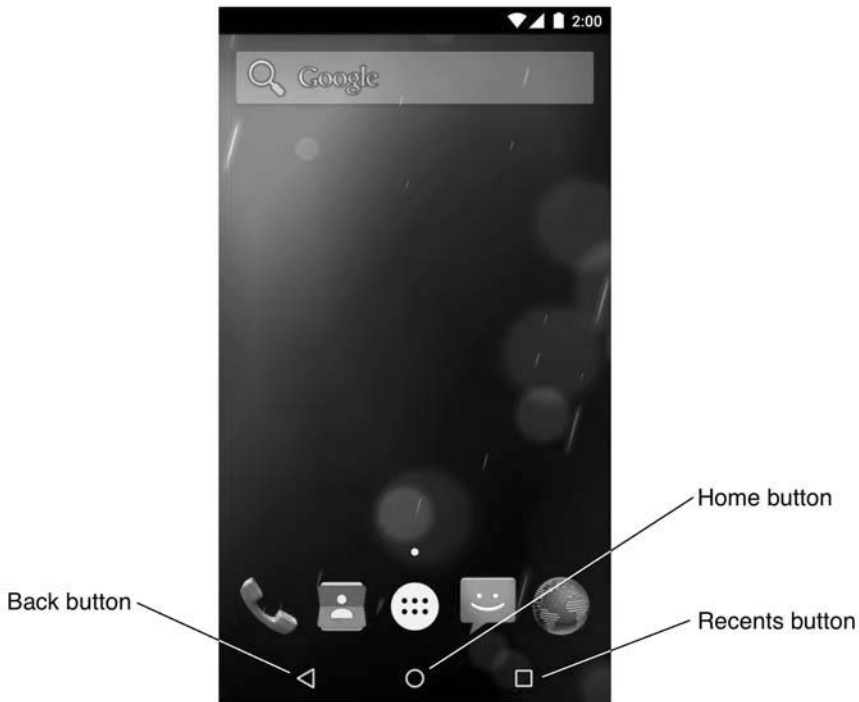
Relaunch GeoQuiz. Press the Home button and then check LogCat. Your activity received calls to **onPause()** and **onStop()**, but not **onDestroy()** (Figure 3.6).

Figure 3.6 Pressing the Home button stops the activity

```
logcat
08-04 17:56:05.477 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 17:56:05.533 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 17:56:05.533 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
08-04 17:56:10.851 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onPause() called
08-04 17:56:11.191 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onStop() called
```

On the device, pull up the task manager: On newer devices, press the Recents button next to the Home button (Figure 3.7). On devices without a Recents button, long-press the Home button.

Figure 3.7 Home, Back, and Recents buttons



In the task manager, press GeoQuiz and then check LogCat. The activity was started and resumed, but it did not need to be created.

Pressing the Home button tells Android, “I’m going to go look at something else, but I might come back.” Android pauses and stops your activity but tries not to destroy it in case you come back.

However, a stopped activity’s survival is not guaranteed. When the system needs to reclaim memory, it will destroy stopped activities.

Another situation that pauses an activity is when it is obscured from the user, such as by a pop-up window. Even if the window only partially covers the activity, the activity is paused and cannot be interacted with. The activity resumes when the pop-up window is dismissed.

As you continue through the book, you will override the different activity lifecycle methods to do real things for your application. When you do, you will learn more about the uses of each method.

Rotation and the Activity Lifecycle

Let’s get back to the bug you found at the end of Chapter 2. Run GeoQuiz, press the Next button to reveal the second question, and then rotate the device. (On the emulator, press Fn+Control+F12/Ctrl+F12 to rotate.)

After rotating, GeoQuiz will display the first question again. Check LogCat to see what has happened. Your output should look like Figure 3.8.

Figure 3.8 **QuizActivity** is dead. Long live **QuizActivity**!

```
logcat
08-04 18:01:32.555 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 18:01:32.555 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 18:01:32.555 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
08-04 18:08:53.557 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onPause() called
08-04 18:08:53.558 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStop() called
08-04 18:08:53.558 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onDestroy() called
08-04 18:08:53.585 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 18:08:53.605 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 18:08:53.605 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
```

When you rotated the device, the instance of **QuizActivity** that you were looking at was destroyed, and a new one was created. Rotate the device again to witness another round of destruction and rebirth.

This is the source of your bug. Each time a new **QuizActivity** is created, `mCurrentIndex` is initialized to 0, and the user starts over at the first question. You will fix this bug in a moment. First, let's take a closer look at why this happens.

Device configurations and alternative resources

Rotating the device changes the *device configuration*. The *device configuration* is a set of characteristics that describe the current state of an individual device. The characteristics that make up the configuration include screen orientation, screen density, screen size, keyboard type, dock mode, language, and more.

Typically, applications provide alternative resources to match different device configurations. You saw an example of this when you added multiple arrow icons to your project for different screen densities.

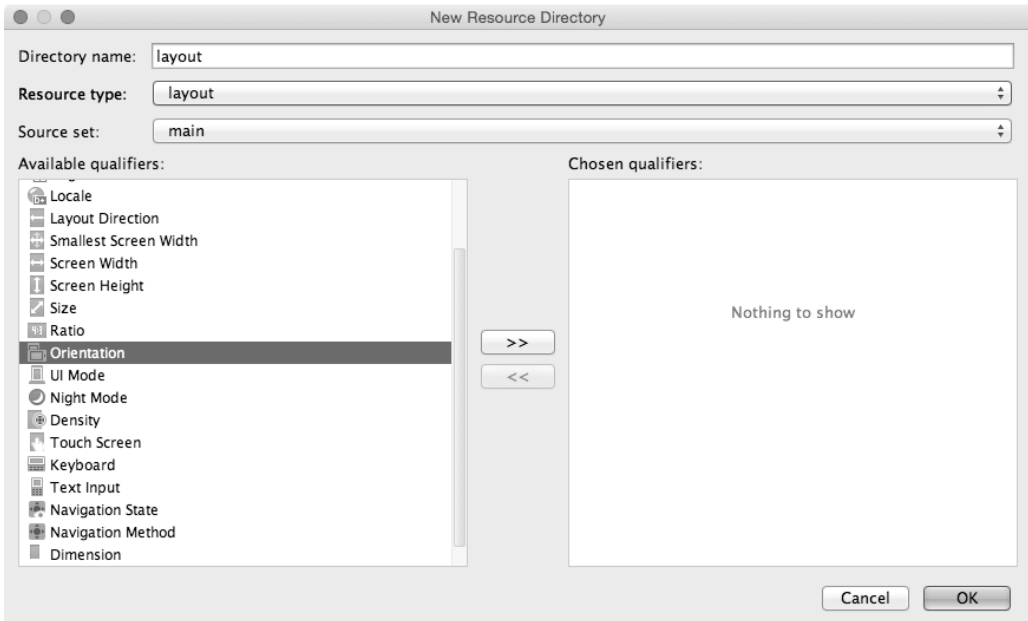
Screen density is a fixed component of the device configuration; it cannot change at runtime. On the other hand, some components, like screen orientation, *can* change at runtime.

When a *runtime configuration change* occurs, there may be resources that are a better match for the new configuration. To see this in action, let's create an alternative resource for Android to find and use when the device's screen orientation changes to landscape.

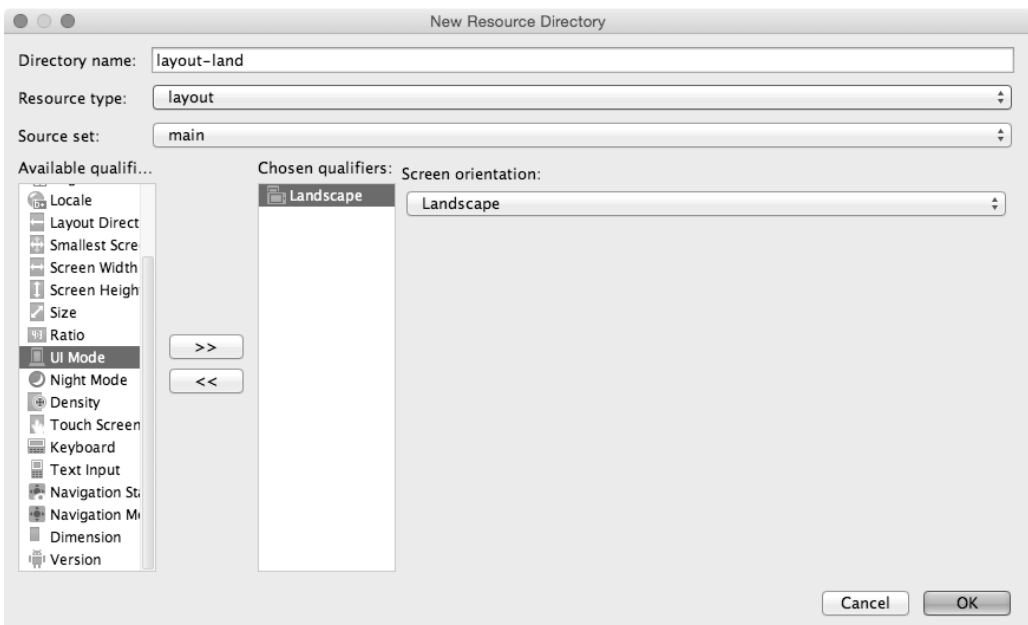
Creating a landscape layout

In the Project tool window, right-click the `res` directory and select `New → Android resource directory`. You should see a window similar to Figure 3.9 that lists the resource types and qualifiers for those types. Select `layout` in the `Resource type` drop-down box. Leave the `Source set` option set to `main`. Next, you will choose how the layout resources will be qualified. Select `Orientation` in the `Available qualifiers` list and click the `>>` button to move `Orientation` to the `Chosen qualifiers` section.

Figure 3.9 Creating a new resource directory



Finally, ensure that Landscape is selected in the Screen Orientation dropdown, as shown in Figure 3.10. Verify that the Directory name now indicates that your directory is called `layout-land`. While this window looks fancy, its purpose is just to set the name of your directory. Click OK and Android Studio will create the `res/layout-land/` folder.

Figure 3.10 Creating `res/layout-land`

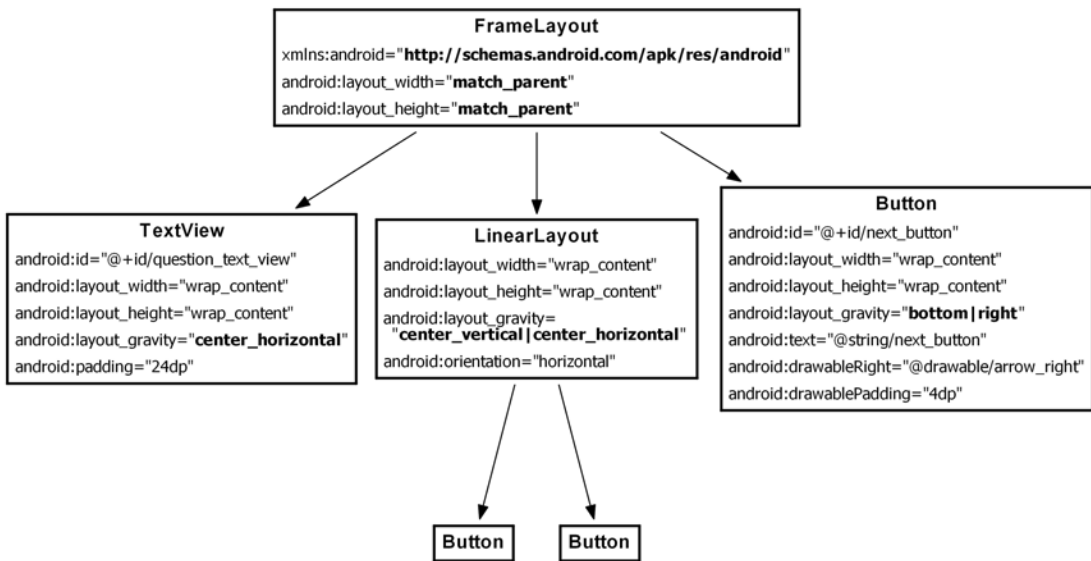
The `-land` suffix is another example of a configuration qualifier. Configuration qualifiers on `res` subdirectories are how Android identifies which resources best match the current device configuration. You can find the list of configuration qualifiers that Android recognizes and the pieces of the device configuration that they refer to at <http://developer.android.com/guide/topics/resources/providing-resources.html>.

When the device is in landscape orientation, Android will find and use resources in the `res/layout-land` directory. Otherwise, it will stick with the default in `res/layout/`. However, at the moment there are no resources in the `res/layout-land` directory. Let's fix that.

Copy the `activity_quiz.xml` file from `res/layout/` to `res/layout-land/`. You now have a landscape layout and a default layout. Keep the filename the same. The two layout files must have the same filename so that they can be referenced with the same resource ID.

Now make some changes to the landscape layout so that it is different from the default. Figure 3.11 shows the changes that you are going to make.

Figure 3.11 An alternative landscape layout



The **FrameLayout** will replace the **LinearLayout**. **FrameLayout** is the simplest **ViewGroup** and does not arrange its children in any particular manner. In this layout, child views will be arranged according to their `android:layout_gravity` attributes.

The **TextView**, **LinearLayout**, and **Button** children of the **FrameLayout** need `android:layout_gravity` attributes. The **Button** children of the **LinearLayout** will stay exactly the same.

Open `layout-land/activity_quiz.xml` and make the necessary changes using Figure 3.11. You can use Listing 3.4 to check your work.

Listing 3.4 Tweaking the landscape layout (layout-land/activity_quiz.xml)

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" →

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

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="24dp" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center_horizontal"
        android:orientation="horizontal" >

        ...

    </LinearLayout>

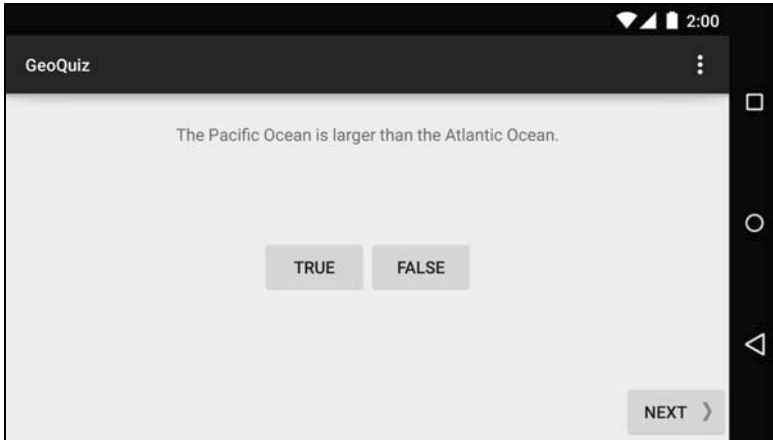
    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp"
        />

</LinearLayout>
</FrameLayout>

```

Run GeoQuiz again. Rotate the device to landscape to see the new layout (Figure 3.12). Of course, this is not just a new layout – it is a new **QuizActivity** as well.

Figure 3.12 QuizActivity in landscape orientation



Rotate back to portrait to see the default layout and yet another new **QuizActivity**.

Android does the work of determining the best resource for you, but it has to create a new activity from scratch to do it. For a **QuizActivity** to display a different layout, **setContentView(R.layout.activity_quiz)** must be called again. And this will not happen unless **QuizActivity.onCreate(...)** is called again. Thus, Android destroys the current **QuizActivity** on rotation and starts fresh to ensure that it has the resources that best match the new configuration.

Note that Android destroys the current activity and creates a new one whenever any runtime configuration change occurs. A change in keyboard availability or language could also occur at runtime, but a change in screen orientation is the runtime change that occurs most frequently.

Saving Data Across Rotation

Android does a great job of providing alternative resources at the right time. However, destroying and re-creating activities on rotation can cause headaches, too, like GeoQuiz's bug of reverting back to the first question when the device is rotated.

To fix this bug, the post-rotation **QuizActivity** needs to know the old value of `mCurrentIndex`. You need a way to save this data across a runtime configuration change, like rotation. One way to do this is to override the **Activity** method:

```
protected void onSaveInstanceState(Bundle outState)
```

This method is normally called by the system before **onPause()**, **onStop()**, and **onDestroy()**.

The default implementation of **onSaveInstanceState(...)** directs all of the activity's views to save their state as data in the **Bundle** object. A **Bundle** is a structure that maps string keys to values of certain limited types.

You have seen this **Bundle** before. It is passed into **onCreate(Bundle)**:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
}
```

When you override **onCreate(...)**, you call **onCreate(...)** on the activity's superclass and pass in the bundle you just received. In the superclass implementation, the saved state of the views is retrieved and used to re-create the activity's view hierarchy.

Overriding onSaveInstanceState(Bundle)

You can override **onSaveInstanceState(...)** to save additional data to the bundle and then read that data back in **onCreate(...)**. This is how you are going to save the value of `mCurrentIndex` across rotation.

First, in `QuizActivity.java`, add a constant that will be the key for the key-value pair that will be stored in the bundle.

Listing 3.5 Adding a key for the value (`QuizActivity.java`)

```
public class QuizActivity extends AppCompatActivity {

    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";

    private Button mTrueButton;
    ...
}
```

Next, override **onSaveInstanceState(...)** to write the value of `mCurrentIndex` to the bundle with the constant as its key.

Listing 3.6 Overriding **onSaveInstanceState(...)** (`QuizActivity.java`)

```
mNextButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
        updateQuestion();
    }
});

updateQuestion();
}

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(TAG, "onSaveInstanceState");
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
```

Finally, in `onCreate(...)`, check for this value. If it exists, assign it to `mCurrentIndex`.

Listing 3.7 Checking bundle in `onCreate(...)` (`QuizActivity.java`)

```
...
if (savedInstanceState != null) {
    mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
}
updateQuestion();
}
```

Run `GeoQuiz` and press `Next`. No matter how many device rotations you perform, the newly minted `QuizActivity` will “remember” what question you were on.

Note that the types that you can save to and restore from a `Bundle` are primitive types and classes that implement the `Serializable` or `Parcelable` interfaces. It is usually a bad practice to put objects of custom types into a `Bundle`, however, because the data might be stale when you get it back out. It is a better choice to use some other kind of storage for the data and put a primitive identifier into the `Bundle` instead.

Testing the implementation of `onSaveInstanceState(...)` is a good idea – especially if you are saving and restoring objects. Rotation is easy to test; testing low-memory situations is harder. There is information at the end of this chapter about how to simulate your activity being destroyed by Android to reclaim memory.

The Activity Lifecycle, Revisited

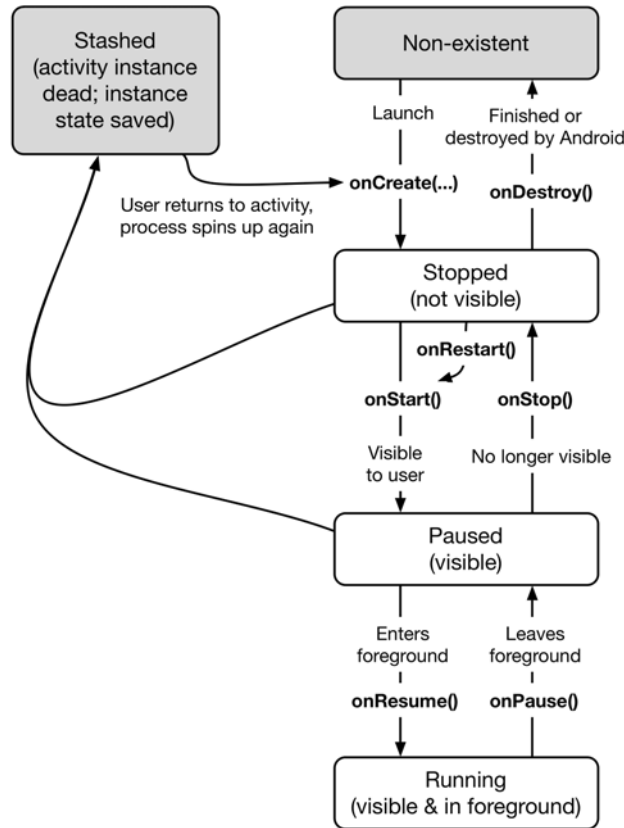
Overriding `onSaveInstanceState(Bundle)` is not just for handling rotation. An activity can also be destroyed if the user navigates away for a while and Android needs to reclaim memory.

Android will never destroy a running activity to reclaim memory – the activity must be in the paused or stopped state to be destroyed. If an activity is paused or stopped, then its `onSaveInstanceState(...)` method has been called.

When `onSaveInstanceState(...)` is called, the data is saved to the `Bundle` object. That `Bundle` object is then stuffed into your activity’s *activity record* by the OS.

To understand the activity record, let’s add a *stashed* state to the activity lifecycle (Figure 3.13).

Figure 3.13 The complete activity lifecycle



When your activity is stashed, an **Activity** object does not exist, but the activity record object lives on in the OS. The OS can reanimate the activity using the activity record when it needs to.

Note that your activity can pass into the stashed state without **onDestroy()** being called. However, you can always rely on **onPause()** and **onSaveInstanceState(...)** to be called. Typically, you override **onSaveInstanceState(...)** to stash small, transient states that belong to the current activity in your **Bundle** and **onPause()** for anything else that needs to be done.

Under some situations, Android will not only kill your activity but also completely shut down your application's process. This will only happen if the user is not currently looking at your application, but it can (and does) happen. Even in this case, the activity record will live on and enable a quick restart of your activity if the user returns.

So when does the activity record get snuffed? When the user presses the Back button, your activity really gets destroyed, once and for all. At that point, your activity record is discarded. Activity records are also typically discarded on reboot and may also be discarded if they are not used for a long time.

For the More Curious: Testing `onSaveInstanceState(Bundle)`

If you are overriding `onSaveInstanceState(Bundle)`, you should test that your state is being saved and restored as expected. This is easy to do on the emulator.

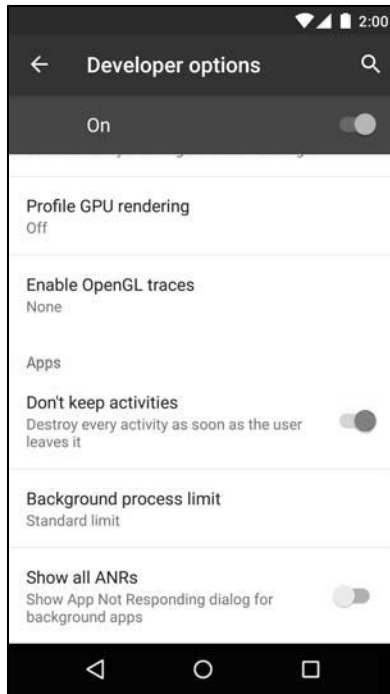
Start up a virtual device. Within the list of applications on the device, find the Settings app (Figure 3.14). This app is included with most system images used on the emulator.

Figure 3.14 Finding the Settings app



Launch Settings and select Developer options. Here you will see many possible settings. Turn on the setting labeled Don't keep activities, as shown in Figure 3.15.

Figure 3.15 Don't keep activities selected



Now run your app and press the Home button. Pressing Home causes the activity to be paused and stopped. Then the stopped activity will be destroyed just as if the Android OS had reclaimed it for its memory. Then you can restore the app to see if your state was saved as you expected. *Be sure to turn this setting off when you are done testing, as it will cause a performance decrease and some apps will perform poorly.*

Pressing the Back button instead of the Home button will always destroy the activity, regardless of whether you have this development setting on. Pressing the Back button tells the OS that the user is done with the activity.

To run the same test on a hardware device, you must install Dev Tools on the device. For more information, visit <http://developer.android.com/tools/debugging/debugging-devtools.html>.

For the More Curious: Logging Levels and Methods

When you use the `android.util.Log` class to send log messages, you control not only the content of a message, but also a *level* that specifies how important the message is. Android supports five log levels, shown in Figure 3.16. Each level has a corresponding method in the `Log` class. Sending output to the log is as simple as calling the corresponding `Log` method.

Figure 3.16 Log levels and methods

Log Level	Method	Notes
ERROR	Log.e(...)	Errors
WARNING	Log.w(...)	Warnings
INFO	Log.i(...)	Informational messages
DEBUG	Log.d(...)	Debug output; may be filtered out
VERBOSE	Log.v(...)	For development only!

In addition, each of the logging methods has two signatures: one which takes a *tag* string and a message string and a second that takes those two arguments plus an instance of **Throwable**, which makes it easy to log information about a particular exception that your application might throw. Listing 3.8 shows some sample log method signatures. Use regular Java string concatenation to assemble your message string, or **String.format** if you have fancier needs.

Listing 3.8 Different ways of logging in Android

```
// Log a message at "debug" log level
Log.d(TAG, "Current question index: " + mCurrentIndex);

Question question;
try {
    question = mQuestionBank[mCurrentIndex];
} catch (ArrayIndexOutOfBoundsException ex) {
    // Log a message at "error" log level, along with an exception stack trace
    Log.e(TAG, "Index was out of bounds", ex);
}
```


Index

Symbols

9-patch images, 376

@+id, 20, 187

@Override, 60

A

aapt (Android Asset Packing tool), 30

action bar, tool bar vs., 254

action view, 455

ACTION_CAPTURE_IMAGE, 299

activities

(see also **Activity**, fragments)

about, 2

abstract fragment-hosting activity, 172

adding to project, 87-109

as controller, 37

back stack of, 107, 108, 393

base, 393

child, 88, 101

creating new, 89

fragment transactions and, 314

handling configuration changes in, 522

hosting fragments, 125, 133-136

label (display name), 388

launcher, 106

lifecycle and fragments, 146

lifecycle diagram, 70

lifecycle of, 57, 63, 64, 70, 71

managing fragments, 314-323

overriding methods, 58

passing data between, 97-106

record, 70

rotation and, 63-68

starting from fragment, 193

starting in current task, 393

starting in new task, 396

states of, 57, 70

tasks and, 393

UI flexibility and, 123

Activity

as **Context** subclass, 24

FragmentActivity, 128

getIntent(), 100, 196

lifecycle methods, 57-63

onActivityResult(...), 103

onCreate(...), 17, 57, 59

onDestroy(), 57

onOptionsItemSelected(MenuItem), 17

onPause(), 57

onResume(), 57, 200

onSaveInstanceState(...), 68-70, 349, 351

onStart(), 57

onStop(), 57

setContentView(...), 17

setResult(...), 103

SingleFragmentActivity, 172, 173, 175, 309

startActivity(...), 95

startActivityForResult(...), 101

activity record, 70

ActivityInfo, 391

ActivityManager

back stack, 107, 108

starting activities, 95, 97, 103

ActivityNotFoundException, 97

Adapter, 179

adapters

defined, 179

implementing, 182

adb (Android Debug Bridge), 46

add(...) (**FragmentTransaction**), 143

addFlags(...) (**Intent**), 396

AlarmManager, 473, 477

AlertDialog, 215, 217, 219, 221

AlertDialog.Builder, 219

constructor, 219

create(), 219

setPositiveButton(...), 219

setTitle(...), 219

setView(...), 221

alias resources, 311-313

ancestral navigation, 248

Android Asset Packing tool, 30

Android Asset Studio, 241

Android Debug Bridge (adb), 46

Android developer documentation, 117, 118

Android firmware versions, 111

Android Lint

as static analyzer, 84

compatibility and, 115-117

running, 84

Android SDK Manager, xxi

Android Studio

- (see also debugging, projects)
- adding dependencies in, 129-132
- AppCompat theme, 358
- build process, 29
- code completion, 25
- code style preferences, 34
- creating new classes, 34
- creating new projects, 2-7
- debugger, 8
 - (see also debugging)
- editor, 8
- Emulator Control, 554
- extracting a method with, 230
- generating getter and setter methods, 34-36
- graphical layout tool, 158
- installing, xxi
- preferences, 34
- project tool window, 8
- project window, 8
- res/values directory, 15
- src directory, 16
- Tool Windows, 8
- views
 - Devices view, 47
 - Lint Warnings view, 84
 - Variables view, 81
- Android versions (see SDK versions)
- Android Virtual Device Manager, 26
- Android Wear, 483
- Android XML namespaces, 13
- android.text.format.DateFormat**, 166
- android.util.Log** (see **Log**)
- android.view.animation**, 550
- android:background**, 354
- android:configChanges**, 522
- android:contentDescription**, 56
- android:documentLaunchMode**, 403
- android:drawablePadding**, 52
- android:drawableRight**, 52
- android:icon**, 241
- android:layout_gravity**, 66
- android:layout_height**, 14
- android:layout_weight**, 164
- android:layout_width**, 14
- android:minSdkVersion**, 114
- android:orientation**, 14
- android:padding**, 157
- android:protectionLevel**, 502
- android:targetSdkVersion**, 113, 114
- AndroidManifest.xml** (see **manifest**)
- animated state list drawables, 590
- animation (see property animation)
- animation tools, 591-597
- AnimatorListener**, 549
- AnimatorSet**, 549
- anonymous inner classes, xix, 22, 23
- API keys
 - maps, 572
 - when working with teams, 584
- API levels (see SDK versions)
- .apk file, 29, 381
- app icon, 249
- app:showAsAction**, 239
- AppCompat library, 216
 - themes in, 358
 - toolbars with, 235-238
- AppCompatActivity**, 17
- appendQueryParameter(...)** (**Uri.Builder**), 417
- application context, 272
- AppTheme**, 357
- arguments bundle, 197-199
- ArrayList**, 170
- AssetManager**, 332, 337
- assets, 325-337
 - accessing, 336
 - importing, 329-331
 - managing, 331-333
 - presenting to user, 333-335
 - vs. resources, 326
- AsyncTask**
 - cancel(...)**, 426
 - doInBackground(...)**, 411
 - for running on background thread, 411
 - vs. **HandlerThread**, 447
 - onPostExecute(...)**, 425
 - onProgressUpdate(...)**, 427
 - publishProgress(...)**, 427
- AsyncTaskLoader**, 427
- AttributeSet**, 530
- auto-complete, 25
- AVDs (Android Virtual Devices)
 - creating, 26
 - for tablets, 307

B

Back button, 61, 393, 524

back stack, 107

background threads

dedicated, 432

updating UI from, 425

using **AsyncTask** for, 411, 415

beginTransaction() (**FragmentTransaction**), 143

Bitmap, 301

BitmapFactory, 301

bitmaps, scaling and displaying, 301-304

breakpoints

(see also debugging)

exception, 82, 83

setting, 79-82

broadcast intents

defined, 491

ordered, 502-507

permissions and, 500, 501

registered in code, 498, 499

regular intents vs., 491

sending, 497

broadcast receivers

defined, 491

dynamic, 498, 499

implementing, 492-494

intent filters and, 492

long-running tasks and, 507

permissions and, 500-502

standalone, 492

uses for, 495, 496

build errors, 85

(see also debugging)

build process, 29

build target, 114

Build.VERSION.SDK_INT, 116

Bundle, 344

for fragment arguments, 197-199

in **onCreate(...)**, 69

in **onSaveInstanceState(...)**, 68

putCharSequence(...); 198

putInt(...); 198

putSerializable(...), 198

Button

adding ID, 20

example, 10

vs. **ImageButton**, 55

inheritance, 55

buttons, 55

9-patch images for, 376

adding icons to, 52

drawables for, 369

floating action, 598, 600

modifying attributes, 365-367

buttonStyle, 365

C

caching, 447

Calendar, 226

Callbacks interface, 314-323

camera, 291-305

firing intent, 299

layouts for, 292-294

taking pictures with intents, 297-300

CameraUpdate, 580

cancel(...) (**AlarmManager**), 477

cancel(...) (**AsyncTask**), 426

Canvas, 536

cards (view component), 597

CheckBox, 151

choosers, creating, 282

circular reveal animation, 591-593

close(), 269

code completion, 25

codenames, version, 111

color

for animation, 546

themes and, 359

colorAccent, 360

colorBackground, 363

colorPrimary, 359

commands (**IntentService**), 468

compatibility

Android Lint and, 115-117

fragments and, 128-132

importance of, xix, 111, 112

issues, 112

minimum SDK version and, 114

with support library, 128-132

using conditional code for, 116

wrapping code for, 114-117

compile SDK version, 114

ComponentName, 392

- components, 96
- concurrent documents, 401-403
- configuration changes, 64, 68, 346
- configuration qualifiers
 - defined, 66
 - for screen density, 49
 - for screen orientation, 66
 - for screen size, 313, 323
- ConnectivityManager**, 470
- contacts
 - getting data from, 285
 - permissions for, 287
- container views, 135, 143
- ContentProvider**, 285
- ContentResolver**, 285
- ContentValues**, 263
- Context**, 272
 - AssetManager** from, 332
 - basic file and directory methods in, 294
 - explicit intents and, 96
 - external file and directory methods in, 295
 - for opening database file, 258
 - getSharedPreferences(...)**, 461
 - resource IDs and, 24
- Context.getExternalFilesDir(String)**, 298
- Context.MODE_WORLD_READABLE, 295
- controller objects, 37
- conventions
 - class naming, 7
 - extra naming, 99
 - package naming, 4
 - variable naming, 21, 34
- create() (AlertDialog.Builder)**, 219
- createChooser(...)** (**Intent**), 282
- Creative Commons, 331
- Cursor**, 267, 269
- CursorWrapper**, 267

- D**
- /data/data directory, 257
- database schema, 258
- databases, SQLite, 257-272
- Date**, 226
- DatePicker**, 221
- debugging
 - (see also Android Lint)
 - build errors, 85
 - crash, 76
 - crash on unconnected device, 77
 - database issues, 261
 - misbehaviors, 77
 - online help for, 86
 - R**, 85
 - running app with debugger, 80
 - stopping debugger, 81
 - when working with teams, 584
- DEFAULT (**Intent**), 397
- delayed execution, 473
- density-independent pixel, 156
- dependencies, adding, 129-132
- dependency injector, 192
- detach(...)** (**FragmentTransaction**), 211
- Dev Tools, 73
- developer documentation, 117, 118
- devices
 - configuration changes and, 64
 - hardware, 26
 - virtual, 26, 307
- Devices view, 47
- Dialog**, 215
- DialogFragment**, 217
 - onCreateDialog(...)**, 219
 - show(...)**, 220
- dialogs, 215-224
- diamond notation, 170
- dip (density-independent pixel), 156
- documentation, 117, 118
- doInBackground(...)** (**AsyncTask**), 411
- dp (density-independent pixel), 156
- draw() (View)**, 536
- draw9patch tool, 379
- drawables, 369
 - 9-patch images, 376
 - for uniform buttons, 369
 - layer list, 374
 - referencing, 52
 - shape, 371
 - state list, 372
- drawing
 - Canvas**, 536
 - in **onDraw(...)**, 536
 - Paint**, 536

E**EditText**, 136

elevation, 589

emulator

(see also virtual devices)

for location testing, 553-556

rotating, 52

running on, 26

search queries on, 460

for tablets, 307

Emulator Control (Android Studio), 554

Environment.getExternalStorageDirectory(...), 295

errors

(see also debugging)

escape sequence (in string), 41

EventBus, 507

exception breakpoints, 82, 83

exceptions, 76, 78

explicit intents

creating, 96

creating at runtime, 392

purpose, 97

external storage

for photos, 294-297

permissions for, 298

extras

defined, 98

fragments retrieving from activity, 195

as key-value pairs, 98

naming, 99

putting, 98, 100

retrieving, 100

structure of, 98

F**File****getCacheDir(...)**, 295**getDir(...)**, 295**getExternalCacheDir(...)**, 296**getExternalFilesDir(...)**, 296**getFilesDir(...)**, 295**FileDescriptor**, 336**FileInputStream**, 295**fileList(...)** (String), 295**FileOutputStream**, 295**File[]****getExternalCacheDirs(...)**, 296**getExternalFilesDirs(...)**, 296**getExternalMediaDirs(...)**, 296

fill_parent, 14

Flickr API, 415

Flickr Geosearch, 563

Flickr, searching in, 449-454

floating action buttons, 598, 600

FloatingActionButton, 598, 600

fluent interface, 143

format string, 278

Fragment

for asset management, 327

getActivity(), 194, 195**getArguments(...)**, 199**getTargetFragment()**, 227**getTargetRequestCode()**, 227

from native libraries, 148

newInstance(...), 198**onActivityResult(...)**, 224**onCreate(Bundle)**, 139**onCreateOptionsMenu(...)**, 244**onCreateView(...)**, 139**onOptionsItemSelected(...)**, 247**onSaveInstanceState(...)**, 139, 349**setArguments(...)**, 198**setHasOptionsMenu(...)**, 244**setRetainInstance(...)**, 345**setTargetFragment(...)**, 226**SingleFragmentActivity**, 328**startActivityForResult(...)**, 203

from support library, 128, 148

fragment arguments, 195, 197-199, 204

fragment transactions, 314, 316

(see also **FragmentTransaction**)**FragmentActivity** (from support library), 128**FragmentManager**

adding fragments to, 142-145

fragment lifecycle and, 145, 146

onResume(), 200

responsibilities, 142

role in rotation, 344, 346

FragmentPagerAdapter, 211

fragments

(see also fragment transactions,

FragmentManager)

about, 123

accessing extra in activity's intent, 195

vs. activities, 123
 activity lifecycle and, 146
 adding in code, 134
 adding to **FragmentManager**, 142-146
 application architecture with, 146
 arguments of, 197-199
 as composable units, 123, 314
Callbacks interface, 314-323
 compatibility and, 128-132
 container views for, 135, 309
 creating, 136
 creating from support library, 138
 delegating functionality to activity, 314
 hosting, 125, 133-136
 implementing lifecycle methods, 139
 inflating layouts for, 139
 layout, 134
 lifecycle diagram, 146
 lifecycle methods, 146
 lifecycle of, 133, 145, 146
 maintaining independence of, 197, 314
 passing data between (same activity), 224
 passing data with fragment arguments, 225
 reasons for, 122-124, 147
 retaining, 345-351
 rotation and, 346-348
 setting listeners in, 140
 starting activities from, 193
 support library and, 128-132, 148
 without support library, 148
 UI flexibility and, 123
 widgets and, 140

FragmentManager, 207
getCount(), 207, 208
getItem(), 207, 208
setOffscreenPageLimit(), 210

FragmentTransaction
add(), 143
beginTransaction(), 143
detach(), 211
remove(), 211

FrameLayout
 as container view for fragments, 135, 309
 described, 66

FusedLocationProviderApi, 564

G

Gallery, 213
 gen directory, 18
 Genymotion, xxii, 554
getAction() (**MotionEvent**), 532
getActiveNetworkInfo()
 (**ConnectivityManager**), 470
getActivity() (**FragmentManager**), 194, 195
getArguments() (**Fragment**), 199
getBackgroundDataSetting()
 (**ConnectivityManager**), 470
getBooleanExtra() (**Intent**), 100
getCacheDir() (**File**), 295
getCount() (**FragmentManager**), 207, 208
getDefaultSharedPreferences()
 (**PreferenceManager**), 461
getDir() (**String name**, **int mode**), 295
getExternalCacheDir() (**File**), 296
getExternalCacheDirs() (**File[]**), 296
getExternalFilesDir() (**String**), 296, 297
getExternalFilesDirs() (**File[]**), 296
getExternalMediaDirs() (**File[]**), 296
getExternalStorageDirectory()
 (**Environment**), 295
getFilesDir() (**File**), 295
getInputStream() (**URLConnection**), 410
getIntent() (**Activity**), 100, 196
getItem() (**FragmentManager**), 207, 208
getMapAsync() (**SupportMapFragment**), 579
getOutputStream() (**URLConnection**), 410
getScaledBitmap(), 301
getSharedPreferences() (**Context**), 461
getTargetFragment() (**Fragment**), 227
getTargetRequestCode() (**Fragment**), 227
 getter and setter methods, generating, 34-36
getTop(), 542
 Google Drive, 402
 Google Play Services
 about, 552
 Maps API from, 571
 setting up, 559
 using, 561-563
GoogleMap, 579
 graphical layout tool, 158-165
GridLayoutManager, 408

GridView, 191

H

Handler, 436, 444

handlers, 436-446

HandlerThread

vs. **AsyncTask**, 447

handling downloads, 433

hardware devices, 26

-hdpi suffix, 49

hero transitions

(see also shared element transitions)

hierarchical navigation, 248

HOME (**Intent**), 397

Home button, 62, 63

home screen, 397, 398

Honeycomb, 112

HTTP networking, 406, 409-411, 414

HttpURLConnection

class, 410

getInputStream(), 410

getOutputStream(), 410

I

icons, 241-243

ImageButton, 55

implicit intents

action, 280, 386

ACTION_CALL category, 289

ACTION_DIAL category, 289

ACTION_PICK category, 283

ACTION_SEND category, 281

benefits of using, 273

categories, 280, 386

CATEGORY_DEFAULT, 387

data, 280

vs. explicit intent, 279

for browsing web content, 514

parts of, 280

sending with **AlarmManager**, 473

taking pictures with, 297-300

type, 280

include, 293

includes, 292, 304

inflating layouts, 17, 139

inheritance, 355, 367

InputStream

for delivering bytes, 410

read(), 410

inSampleSize, 302

insert(...), 264

Intent

addFlags(...), 396

constructors, 96

createChooser(...), 282

getBooleanExtra(...), 100

putExtra(...), 98, 195

setClassName(...), 392

setComponent(...), 392

intent filters

BOOT_COMPLETED, 493

explained, 280

MAIN, 106

SHOW_NOTIFICATION, 498

intent services

processing commands, 468

purpose, 467

Intent.FLAG_ACTIVITY_NEW_DOCUMENT, 403

intents

(see also broadcast intents, explicit intents,

extras, implicit intents, **Intent**)

communicating with, 96, 97

implicit vs. explicit, 97, 273

regular vs. broadcast, 491

taking pictures with, 297-300

IntentService, 467

interpolators, 546

invalidate() (View), 535

is prefix for variable names, 34

J

JavaScript Object Notation (see JSON)

JavaScript, enabling, 518

Javascript, injecting objects, 523

JobScheduler, 486

JobService, 486

JSON (JavaScript Object Notation), 415

JSONObject, 419

L

-land qualifier, 66

LatLngBounds, 580

launcher activities, 106

LAUNCHER category, 106, 387

- layer-list drawables, 374
 - layout attributes
 - android:background, 354
 - android:contentDescription, 56
 - android:drawablePadding, 52
 - android:drawableRight, 52
 - android:icon, 241
 - android:layout_gravity, 66
 - android:layout_height, 14
 - android:layout_weight, 164
 - android:layout_width, 14
 - android:orientation, 14
 - android:padding, 157
 - layout parameters (layout_), 157
 - LayoutInflater**, 30, 140
 - LayoutManager**, 327
 - layouts
 - (see also graphical layout tool, layout attributes, widgets)
 - about, 2
 - alternative, 64-67
 - for asset management, 326
 - for cameras, 292-294
 - defining in XML, 12-14
 - design documentation, 156
 - inflating, 17, 139
 - landscape, 64-67
 - managing multiple, 166
 - naming, 7
 - previewing, 15, 91
 - for property animation, 540
 - root element, 13
 - view hierarchy and, 13
 - layout_weight attribute, 164
 - ldpi suffix, 49
 - LinearLayout**, 10, 13
 - lint (see Android Lint)
 - Lint Warnings view, 84
 - List**, 170
 - list items
 - customizing display of, 185
 - list(String)**, 332
 - list-detail interfaces, 121, 205, 307-317
 - listeners
 - defined, 22
 - as interfaces, 22
 - setting in fragments, 140
 - setting up, 22-25
 - lists
 - displaying, 167
 - getting item data, 179
 - ListView**, 191
 - load(Sound)**, 340
 - Loader**, 427
 - LoaderManager**, 427
 - loadLabel(...)** (**ResolveInfo**), 388
 - local files, 257
 - local layout rect, 542
 - LocalBroadcastManager**, 507, 509
 - location, 551-568
 - adding GPS permissions for, 560
 - finding and displaying images related to, 566
 - with Flickr Geosearch, 563
 - testing, 553-556
 - Location API, 552
 - LocationListener**, 565
 - LocationRequest**, 564
 - Log**, 58
 - Log.d(...)**, 78
 - LogCat
 - (see also logging)
 - logging
 - of exceptions, 78
 - levels, 73
 - Log.d(...)**, 78
 - messages, 58
 - methods, 73
 - of stack traces, 78
 - TAG constant, 58
 - Looper**, 433, 436
 - LRU (least recently used) caching strategy, 447
 - LRUCache**, 447
- ## M
- m prefix for variable names, 21
 - MAIN category, 106, 387
 - main thread, 413
 - makeText(...)** (**Toast**), 24
 - manifest
 - (see also manifest attributes)
 - about, 92
 - adding network permissions to, 411
 - adding service to, 468
 - adding uses-permission INTERNET, 411
 - Android versions in, 113

- build process and, 29
- declaring **Activity** in, 92
- uses-sdk, 113
- manifest attributes
 - android:configChanges, 522
 - android:protectionLevel, 502
- MapFragment**, 574
- maps, 571-584
 - adding markers to, 582
 - API setup for, 572-574
 - getting lat-lon data for, 576-579
 - working with, 579-581
- Maps API, 572-574
- Maps API key, 572
- MapView**, 574
- master-detail interfaces, 121, 205, 307-317
- match_parent, 14
- material design, 587-601
 - animation tools, 591-597
 - material surfaces, 587-591
 - view components, 597-601
- mContext**, 272
- mdpi suffix, 49
- MediaStore**, 298, 299
- MediaStore.ACTION_IMAGE_CAPTURE, 298
- MediaStore.EXTRA_OUTPUT, 299
- MenuItem**, 247
- menus
 - (see also toolbar)
 - about, 238
 - app:showAsAction, 239
 - creating, 244
 - creating XML file for, 239
 - defining in XML, 239
 - determining selected item, 247
 - populating with items, 244
 - as resources, 239
 - responding to selections, 246
- Message**, 436
- message handlers, 436-446
- message loop, 432
- message queue, 432
- messages, 435-446
- minimum required SDK, 113
- minSdkVersion, 114
- mipmap images, 381
- model layer, 37
- model objects, 37

- model objects, from databases, 269-271
- motion events, handling, 532-535
- MotionEvent**
 - actions, 532
 - class, 532
 - getAction()**, 532
- mSoundId, 340
- mSoundPool.load(...)**, 341
- MVC (Model-View-Controller), 37, 38

N

- namespace, Android XML, 13
- navigation, 248
- network, checking availability of, 470
- networking (HTTP), 406, 409, 410, 414
- networking permissions, 411
- NetworkOnMainThreadException**, 413
- newInstance(...)** (**Fragment**), 198
- 9-patch images, 376
- Notification**, 481
- NotificationManager**, 481
- notifications, 481-483
- notify(...)** (**NotificationManager**), 481
- NullPointerException**, 77

O

- ObjectAnimator**, 543
- onActivityResult(...)** (**Activity**), 103
- onActivityResult(...)** (**Fragment**), 224
- onBindViewHolder**, 183
- OnCheckedChangeListener** interface, 154
- onClick(View)** (**OnClickListener**), 23
- OnClickListener** interface, 22
- onCreate(Bundle)** (**Activity**), 17, 57
- onCreate(...)** (**Fragment**), 139
- onCreateDialog(...)** (**DialogFragment**), 219
- onCreateOptionsMenu(...)** (**Action**), 244
- onCreateOptionsMenu(...)** (**Fragment**), 244
- onCreateView(...)** (**Fragment**), 139
- onCreateViewHolder(...)**, 183, 430
- onDestroy()** (**Activity**), 57
- onDraw(...)** (**View**), 536
- onOptionsItemSelected** (**MenuItem**), 17
- onOptionsItemSelected(...)** (**Fragment**), 247
- onPause()** (**Activity**), 57
- onPostExecute(...)** (**AsyncTask**), 425
- onProgressChanged(...)** (**WebChromeClient**), 521

onProgressUpdate(...) (**AsyncTask**), 427
OnQueryTextListener(...) (**SearchView**), 458
onReceivedTitle(...) (**WebChromeClient**), 521
onRestoreInstanceState(...) (**View**), 538
onResume() (**Activity**), 57, 200
onResume() (**FragmentManager**), 200
onSaveInstanceState(...) (**Activity**), 68-73
onSaveInstanceState(...) (**Activity**), 349, 351
onSaveInstanceState(...) (**Fragment**), 139, 349
onSaveStateInstance() (**View**), 538
onStart() (**Activity**), 57
onStop() (**Activity**), 57
onTextChanged(...) (**TextWatcher**), 141
onTouchEvent(...) (**View**), 532
OnTouchListener (**View**), 532
openConnection() (**URL**), 410
openFileInput(...) (**FileInputStream**), 295
openFileOutput(...) (**FileInputStream**), 295
openNonAssetFd(...), 337
options objects, 582
overflow menu, 239
@Override, 60
overview screen, 393

P

PackageManager, 299
 class, 287
 resolveActivity(...), 287
packages, naming, 4
padding, 157
Paint, 536
Parcelable, 344, 538
parent, 356, 367
PendingIntent, 477
permissions, 411
permissions (defined), 298
persistent data, 460-464
photos
 designating file location for, 296
 external storage, 294-297
 scaling and displaying bitmaps, 301-304
 taking with intents, 297-300
PhotoView, 302
Play Services (see Google Play Services)
play(Sound), 341
PointF, 533
post(...) (**Handler**), 444

preferences (Android Studio), 34
preloading, 447
presses, responding to, 190
processes, 398, 401
progress indicator
 hiding, 521
 updating from background thread, 427
projects
 adding resources, 49
 configure, 5
 creating, 2-7
 gen directory, 18
 layout, 7
 res/layout directory, 18
 res/menu directory, 239
 res/values directory, 18
 setting package name, 3
 setting project name, 3
 src directory, 16
property animation, 539-550
 building scene for, 539
 running multiple animators, 548
 simple, 542-548
protection level values, 502
publishProgress(...) (**AsyncTask**), 427
putCharSequence(...); (**Bundle**), 198
putExtra(...) (**Intent**), 195
putInt(...); (**Bundle**), 198
putSerializable(...) (**Bundle**), 198

Q

query(...), 266

R

R, 18
randomUUID(), 132
read() (**InputStream**), 410
Recents button, 62
RecyclerView, 176-184, 326
 efficient reloading of, 203
 for display grid, 408
 setOnItemClickListener(...), 514
RelativeLayout, 186, 187
release key, 29
remove(...) (**FragmentManager**), 211
request code (**Activity**), 101
res/layout directory, 18

- res/menu directory, 239
 - res/values directory, 15, 18, 353
 - resolveActivity(...)** (**PackageManager**), 287, 304
 - ResolveInfo**, 388
 - resource IDs, 18-20
 - + prefix in, 20, 187
 - multiple layouts and, 166
 - syntax, 187
 - resources
 - (see also configuration qualifiers, drawables, layouts, menus, string resources)
 - about, 18
 - adding, 49
 - alias, 311-313
 - vs. assets, 326
 - location of, 18
 - referencing in XML, 52
 - string, 14, 15
 - result code (**Activity**), 102
 - retained fragments, 345-351
 - retainInstance property (**Fragment**), 345, 346
 - rotation
 - activity lifecycle and, 63-68
 - onSaveInstanceState(...)** and, 349, 351
 - saving data across, 68-70
 - with **DatePicker**, 223
 - rows, inserting and updating, 264
 - running on device, 46-48
 - RxJava, 508
- S**
- s prefix for variable names, 34
 - sandbox, 257
 - savedInstanceState, 345
 - scale-independent pixel, 156
 - schema, database, 258
 - screen orientation, 66
 - screen pixel density, 49, 155
 - screen size, determining, 323
 - SD card, 296
 - SDK versions
 - (see also compatibility)
 - build target, 114
 - codenames, 111
 - installing, xxi
 - listed, 111
 - minimum required, 113
 - target, 113
 - updating, xxii
 - search, 449-465
 - in Flickr, 449-454
 - integrating into app, 449
 - user-initiated, 455-460
 - SearchView**, 455-460
 - bug, 460
 - OnQueryTextListener(...)**, 458
 - post Honeycomb, 456
 - responding to user interactions, 458
 - Serializable**, 344
 - services
 - adding to manifest, 468
 - bound, 485
 - lifecycle of, 484
 - locally bound, 485
 - non-sticky, 484
 - notifying user, 481
 - purpose of, 467
 - remotely bound, 486
 - sticky, 484
 - setArguments(...)** (**Fragment**), 198
 - setClassName(...)** (**Intent**), 392
 - setComponent(...)** (**Intent**), 392
 - setContentView(...)** (**Activity**), 17
 - setHasOptionsMenu(...)** (**Fragment**), 244
 - setInexactRepeating(...)** (**AlarmManager**), 476
 - setJavaScriptEnabled(...)** (**WebSettings**), 519
 - setOffscreenPageLimit(...)** (**FragmentStatePagerAdapter**), 210
 - setOnClickListener(...)**, 22
 - setOnItemClickListener(...)** (**RecyclerView**), 514
 - setOnTouchListener(...)** (**View**), 532
 - setPositiveButton(...)** (**AlertDialog.Builder**), 219
 - setRepeating(...)** (**AlarmManager**), 476
 - setResult(...)** (**Activity**), 102, 103, 203
 - setRetainInstance(...)** (**Fragment**), 345
 - setTargetFragment(...)** (**Fragment**), 226
 - setter methods, generating, 34-36
 - setText(...)** (**TextView**), 101
 - setTitle(...)** (**AlertDialog.Builder**), 219
 - setView(...)** (**AlertDialog.Builder**), 221
 - shape drawables, 371
 - ShapeDrawable**, 371

- shared element transitions, 593-597
 - SharedPreferences**, 461
 - shouldOverrideUrlLoading(...)** (**WebViewClient**), 519
 - show()** (**Toast**), 24
 - show(...)** (**DialogFragment**), 220
 - signing key, 572
 - simulator (see emulator)
 - SingleFragmentActivity**, 172, 173, 175, 309, 328, 384
 - singletons, 168, 192
 - snackbars, 600, 601
 - solutions file, 48
 - Sound**, 340
 - SoundPool**, 339-345
 - audio playback, 341-344
 - creating, 339
 - load(Sound)**, 340
 - loading sounds into, 340
 - mSoundPool.load(...)**, 341
 - play(Sound)**, 341
 - rotation and object continuity with, 344
 - SoundPool.play(...)**, 342
 - SoundPool.release()**, 343
 - unloading sounds, 343
 - SoundPool.play(...)**, 342
 - SoundPool.release()**, 343
 - sp (scale-independent pixel), 156
 - SQLite databases, 257-272
 - building, 258-262
 - debugging, 261
 - defining schema for, 258
 - inserting and updating rows, 264
 - model objects from, 269-271
 - reading from, 266-271
 - writing to, 263-266
 - SQLiteDatabase.query(...)**, 266
 - src directory, 16
 - stack traces
 - in LogCat, 76
 - logging of, 78
 - startActivity(...)** (**Activity**), 95
 - startActivityForResult(...)** (**Activity**), 101
 - startActivityForResult(...)** (**Fragment**), 203
 - stashable objects, 344
 - state list animators, 590
 - state list drawables, 372
 - STREAM_MUSIC, 340
 - string resources
 - creating, 15
 - defined, 14
 - referencing, 52
 - String.replace(...)**, 334
 - String.split(...)**, 334
 - strings file, 14, 15
 - strings.xml, 15
 - String[]**, 295
 - styles, 354-356
 - defined, 154
 - inheritance, 355, 367
 - modifying button attributes, 365-367
 - themes and, 154
 - support library, 128-132, 148
 - SupportMapFragment**, 574
 - SupportMapFragment.getMapAsync(...)**, 579
 - sw600dp suffix, 313
 - sync adapters, 488
 - system icons, 241-243
- ## T
- tables, creating, 261
 - tablets
 - creating virtual devices for, 307
 - user interfaces for, 307-317
 - TAG constant, 58
 - target fragment, 226
 - target SDK version, 113
 - targetSdkVersion**, 113, 114
 - task manager, 62
 - tasks
 - and Back button, 393
 - defined, 393
 - vs. processes, 398, 401
 - starting new, 395-397
 - switching between, 393
 - temporal navigation, 248
 - TextView**
 - and **tools:text**, 91
 - example, 10
 - inheritance, 55
 - setText(...)**, 101
 - TextWatcher** interface, 141
 - theme**, 357
 - Theme.AppCompat**, 361
 - themes, 154, 357-364

- accessing attributes, 368
- adding colors to, 359
- modifying, 357
- overriding attributes, 360-364
- threads
 - background (see background threads)
 - main, 413
 - message queue, 432
 - processes and, 399
 - as sequence of execution, 413
 - UI, 413

TimeInterpolator, 546

tinting, 360

Toast, 24

toasts, 23-25

toolbar

- action bar vs., 254
- action view in, 455
- `app:showAsAction`, 239
- features, 235
- menu, 238
- overflow menu, 239

Toolbar

- `onCreateOptionsMenu(...)`, 244

touch events, handling, 532-535

transformation properties, 544

transitions, animation, 550

TypeEvaluator, 548

U

UI fragments (see fragments)

UI thread, 414

Up button, 248, 249

`update(...)`, 265

Uri, 299

Uri.Builder, 417

URL

- for making URL from string, 410
- `openConnection()`, 410

URLConnection, 410

user interfaces

- defined by layout, 2
- for tablets, 307-317
- laying out, 9-16

uses-sdk, 113

UUID.randomUUID(), 132

V

variable names

- conventions for, 34
- prefixes for, 34

Variables view, 81

versions (Android SDK) (see SDK versions)

versions (firmware), 111

View

(see also views, widgets)

`draw()`, 536

`invalidate()`, 535

OnClickListener interface, 22

`onDraw(...)`, 536

`onRestoreStateInstance(...)`, 538

`onSaveStateInstance()`, 538

`onTouchEvent(...)`, 532

`setOnTouchListener(...)`, 532

subclasses, 9, 55

view components, 597-601

view layer, 37

view objects, 37

ViewGroup, 13, 66

ViewHolder, 177, 182, 388

ViewPager, 205-213

- in support library, 207

- internals of, 212

views

- creating, 530

- creation by **RecyclerView**, 177

- custom, 530-532

- laying out in code, 213

- persisting, 538

- simple vs. composite, 530

- touch events and, 532-535

- using fully qualified name in layout, 531

ViewTreeObserver, 305

virtual devices

- (see also emulator)

- for tablets, 307

- testing low-memory handling, 72

W

web content

- browsing via implicit intent, 514

- displaying within an activity, 516

- enabling JavaScript, 518

web rendering events, responding to, 519

WebChromeClient

- for enhancing appearance of **WebView**, 520
- interface, 520
- onProgressChanged(...)**, 521
- onReceivedTitle(...)**, 521

WebSettings, 519

WebView

- for presenting web content, 516
- handling rotation, 522

WebViewClient, 519

widgets

- about, 9
- attributes of, 12, 157
- Button**, 10, 55
- CheckBox**, 151
- DatePicker**, 221
- defining in XML, 12-14
- EditText**, 136
- FrameLayout**, 66
- ImageButton**, 55
- LinearLayout**, 10, 13
- padding, 157
- references, 21
- styles and, 354
- TextView**, 10, 91
- in view hierarchy, 13
- as view layer, 37
- wiring in fragments, 140
- wiring up, 21

wrap_content, 14

X

-xhdpi suffix, 49

XML

- Android namespace, 13
- referencing resources in, 52

XML drawables (see drawables)

-xxhdpi suffix, 49

Z

Z values, 589