Dedication

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

— B.H.
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.

- Chris Stewart and Owen Matthews, who contributed great foundational content for several chapters.

- Our co-instructors, Chris Stewart and Christopher Moore. We thank them for their patience in teaching work-in-progress material, their suggestions and corrections for that material, and their consultation when we were considering sweeping changes.

- Our coworkers Bolot Kerimbaev and Andrew Lunsford. Their feedback was instrumental in our decision to highlight the use of fragments.

- Our technical reviewers, Frank Robles, Jim Steele, Laura Cassell, Mark Dalrymple, and Magnus Dahl, 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, Susan Loper, has an amazing ability to turn our programmatic ramblings and bad jokes into thoughtful, concise prose. And better jokes. Without her help, this would not have been a fun book to read. She taught us everything we know about clear and approachable technical writing.

- Thanks to NASA. Our little book seems small and silly in comparison to exploring the solar system.

- Ellie Volckhausen, who designed our cover.

- Elizabeth Holaday, 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.

- The folks at Facebook, who gave us so much fantastic feedback on the course.

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  
How to Use This Book .................................................. xvii  
How This Book Is Organized .......................................... xviii  
  Challenges .......................................................................... xviii  
  Are you more curious? ...................................................... xix  
Code Style ............................................................................... xix  
Typographical Conventions .......................................... xix  
Android Versions ............................................................ xx  
The Necessary Tools .......................................................... xx  
  Downloading and installing the ADT Bundle .................. xx  
  Downloading earlier SDK versions .............................. xxi  
A hardware device ............................................................. xxi  
1. Your First Android Application ....................................... 1  
  App Basics .......................................................................... 2  
  Creating an Android Project .......................................... 3  
  Navigating in Eclipse ....................................................... 7  
Laying Out the User Interface .............................................. 8  
  The view hierarchy .......................................................... 12  
  Widget attributes ............................................................ 12  
  Creating string resources .............................................. 14  
  Previewing the layout ...................................................... 14  
From Layout XML to View Objects .................................. 15  
  Resources and resource IDs .......................................... 16  
Wiring Up Widgets ............................................................ 18  
  Organizing imports .......................................................... 19  
  Getting references to widgets ...................................... 20  
  Setting listeners .............................................................. 20  
Running on the Emulator .................................................. 24  
For the More Curious: Android Build Process .................. 26  
  Android build tools .......................................................... 28  
2. Android and Model-View-Controller ................................. 31  
  Creating a New Class .......................................................... 32  
  Generating getters and setters ...................................... 33  
Model-View-Controller and Android ............................. 35  
  Benefits of MVC .............................................................. 36  
Updating the View Layer .................................................. 37  
Updating the Controller Layer ....................................... 39  
Running on a Device .......................................................... 44  
  Connecting your device .................................................. 44  
  Configuring your device for development .................... 45  
Adding an Icon ................................................................. 45  
  Adding resources to a project ...................................... 46  
  Referencing resources in XML .................................... 47  
Challenges ............................................................................. 48
7. UI Fragments and the Fragment Manager ................................................................. 125
   The Need for UI Flexibility .................................................................................. 126
   Introducing Fragments ....................................................................................... 126
   Starting CriminalIntent ....................................................................................... 127
      Creating a new project .................................................................................. 130
      Fragments and the support library .................................................................. 131
      Creating the Crime class .............................................................................. 133
   Hosting a UI Fragment ....................................................................................... 134
      The fragment lifecycle .................................................................................. 134
      Two approaches to hosting .......................................................................... 135
      Defining a container view ............................................................................ 136
   Creating a UI Fragment .................................................................................... 137
      Defining CrimeFragment’s layout .................................................................. 137
      Creating the CrimeFragment class .................................................................. 139
   Adding a UI Fragment to the FragmentManager ................................................. 142
      Fragment transactions .................................................................................. 143
      The FragmentManager and the fragment lifecycle ....................................... 145
   The Reason All Our Activities Will Use Fragments ............................................ 147
   For the More Curious: Developing for Honeycomb, ICS, Jelly Bean, and Beyond .................................................................................. 147
8. Creating User Interfaces with Layouts and Widgets ........................................... 149
   Upgrading Crime ............................................................................................... 149
   Updating the Layout .......................................................................................... 150
   Wiring Widgets .................................................................................................. 152
   More on XML Layout Attributes ........................................................................ 153
      Styles, themes, and theme attributes .............................................................. 153
      Screen pixel densities and dp and sp ............................................................ 154
      Android’s design guidelines ......................................................................... 155
      Layout parameters ....................................................................................... 156
      Margins vs. padding ..................................................................................... 156
   Using the Graphical Layout Tool ...................................................................... 157
      Adding a new widget .................................................................................... 159
      Editing attributes in properties .................................................................... 160
      Reorganizing widgets in the outline view ...................................................... 161
      Updating child layout parameters ................................................................ 162
      How android:layout_weight works .............................................................. 164
      Summary of graphical layout tool .................................................................. 165
      Widget IDs and multiple layouts .................................................................... 165
   Challenge: Formatting the Date ........................................................................ 166
9. Displaying Lists with ListFragment .................................................................... 167
   Updating CriminalIntent’s Model Layer ............................................................. 168
   Singletons and centralized data storage ............................................................ 169
   Creating a ListFragment .................................................................................... 170
   An Abstract Activity for Hosting a Fragment ..................................................... 171
      A generic fragment-hosting layout ............................................................... 172
      An abstract Activity class ............................................................................ 172
   ListFragment, ListView, and ArrayAdapter ...................................................... 178
   Creating an ArrayAdapter<T> ......................................................................... 180
   Responding to list item clicks ......................................................................... 183
Android Programming

Default resources .................................................................................................................. 246
Configuration Qualifiers ..................................................................................................... 247
Prioritizing alternative resources ........................................................................................ 247
Multiple qualifiers ............................................................................................................... 249
Finding the best-matching resources .................................................................................. 250
More Resource Rules and Regulations ............................................................................... 251
Resource naming .................................................................................................................. 251
Resource directory structure .............................................................................................. 251
Testing Alternative Resources ............................................................................................ 251

16. The Action Bar .................................................................................................................. 253

Options Menus .................................................................................................................... 253
Defining an options menu in XML ....................................................................................... 255
Creating the options menu .................................................................................................. 257
Responding to options menu selections ............................................................................. 260
Enabling Ancestral Navigation ............................................................................................ 262
Enabling the app icon ......................................................................................................... 262
Responding to the Up button ............................................................................................... 264
An Alternative Menu Item .................................................................................................. 266
Creating an alternative menu file ....................................................................................... 266
Toggling the menu item title ............................................................................................... 267
“Just one more thing...” ..................................................................................................... 268
Chalenge: An Empty View for the List ............................................................................... 270

17. Saving and Loading Local Files ...................................................................................... 273

Saving and Loading Data in CriminalIntent ....................................................................... 273
 Saving crime data to a JSON file ....................................................................................... 274
 Loading crimes from the filesystem ................................................................................... 278
Challenge: Use External Storage ........................................................................................ 280
For the More Curious: The Android Filesystem and Java I/O ........................................... 280
 Accessing files and directories ........................................................................................ 281

18. Context Menus and Contextual Action Mode ................................................................ 283

Defining a Context Menu Resource .................................................................................. 284
Implementing a Floating Context Menu ........................................................................... 284
Creating the context menu ................................................................................................... 284
Registering for the context menu ......................................................................................... 285
Responding to an action ....................................................................................................... 286
Implementing Contextual Action Mode ............................................................................... 287
Enabling multiple selection ................................................................................................. 288
Action mode callbacks in a list view ................................................................................... 289
Changing activated item backgrounds ................................................................................ 291
Implementing contextual action mode in other views ....................................................... 293
Compatibility: Fallback or Duplicate? ............................................................................... 293
Challenge: Deleting from CrimeFragment .......................................................................... 294
For the More Curious: ActionBarSherlock ........................................................................ 294
Challenge: Using ActionBarSherlock ................................................................................ 296
Basic ABS integration in CriminalIntent ........................................................................... 297
More advanced integration ................................................................................................. 297
Even more advanced integration ......................................................................................... 298

19. Camera I: Viewfinder ..................................................................................................... 299
Creating the Fragment Layout ................................................................................. 300
Creating CrimeCameraFragment .............................................................................. 302
Creating CrimeCameraActivity ................................................................................ 303
Adding activity and camera permissions to the manifest ........................................ 303
Using the Camera API ........................................................................................... 304
Opening and releasing the camera .................................................................... 304
SurfaceView, SurfaceHolder, and Surface ........................................................... 306
Determining preview size ............................................................................... 310
Starting CrimeCameraActivity from CrimeFragment ............................................ 311
For the More Curious: Running Activities from the Command Line ......................... 316
20. Camera II: Taking Pictures and Handling Images ........................................................... 319
Taking a Picture .................................................................................................... 319
Implementing Camera callbacks ....................................................................... 321
Setting the picture size ................................................................................... 324
Passing Data Back to CrimeFragment ................................................................ 324
Starting CrimeCameraActivity for a result ....................................................... 325
Setting a result in CrimeCameraFragment ......................................................... 326
Retrieving filename in CrimeFragment ................................................................ 327
Updating the Model Layer .................................................................................... 328
Adding a Photo class ..................................................................................... 329
Giving Crime a photo property ........................................................................ 330
Setting the photo property ............................................................................... 330
Updating CrimeFragment’s View ........................................................................ 331
Adding an ImageView .................................................................................... 332
Image handling ............................................................................................. 334
Showing Larger Image in a DialogFragment ...................................................... 338
Challenge: Crime Image Orientation .................................................................... 341
Challenge: Deleting Photos .................................................................................. 341
For the More Curious: Deprecation in Android ...................................................... 342
21. Implicit Intents ........................................................................................................ 345
Adding Buttons ...................................................................................................... 346
Adding a Suspect to the Model Layer .................................................................... 348
Using a Format String ............................................................................................ 348
Using Implicit Intents ........................................................................................... 350
Parts of an implicit intent .................................................................................... 350
Sending a crime report ....................................................................................... 351
Asking Android for a contact ........................................................................... 353
Checking for responding activities ..................................................................... 357
Challenge: Another Implicit Intent ....................................................................... 357
22. Two-Pane Master-Detail Interfaces .............................................................................. 359
Adding Layout Flexibility ....................................................................................... 360
Modifying SingleFragmentActivity ....................................................................... 361
Creating a layout with two fragment containers .................................................. 361
Using an alias resource .................................................................................... 363
Activity: Fragment Boss ......................................................................................... 365
Fragment callback interfaces .................................................................................. 365
For the More Curious: More on Determining Device Size ............................................. 373
23. More About Intents and Tasks ................................................................................... 375
Creating a search interface ................................................................. 453
Searchable activities ........................................................................... 455
Hardware search button .................................................................... 458
How Search works ............................................................................ 458
Launch modes and new intents ......................................................... 459
Simple persistence with shared preferences ................................. 461
Using SearchView on Post-Android 3.0 ............................................. 463
Challenges ....................................................................................... 465

29. Background Services ................................................................................................ 467
Creating an IntentService ............................................................... 467
What Services Are For ....................................................................... 470
Safe background networking ............................................................ 470
Looking for New Results ............................................................... 471
Delayed Execution with AlarmManager ......................................... 472
PendingIntent ................................................................................ 474
Managing alarms with PendingIntent ............................................ 475
Controlling Your Alarm ................................................................. 475
Updating options menu items ....................................................... 477
Notifications .................................................................................. 478
For the More Curious: Service Details ........................................... 480
What service does (and does not) do ............................................. 480
A service’s lifecycle ......................................................................... 480
Non-sticky services ....................................................................... 481
Sticky services ............................................................................. 481
Bound services ............................................................................. 481

30. Broadcast Intents ..................................................................................................... 485
Waking Up on Boot ........................................................................ 485
Broadcast receivers in the manifest .............................................. 486
How to use receivers ....................................................................... 488
Filtering Foreground Notifications ................................................ 489
Sending broadcast intents .............................................................. 489
Dynamic broadcast receivers ......................................................... 490
Using private permissions ............................................................... 492
Receiving results with ordered broadcasts ................................. 495
Receivers and Long-running Tasks ............................................... 499

31. Browsing The Web & WebView ............................................................................... 501
One Last Bit of Flickr Data ............................................................. 501
The Easy Way: Implicit Intents ....................................................... 503
The Harder Way: WebView ............................................................ 503
Using WebChromeClient to spruce things up .............................. 508
Proper rotation with WebView ......................................................... 509
For the More Curious: Injecting JavaScript Objects ..................... 510

32. Custom Views and Touch Events ........................................................................ 513
Setting Up the DragAndDraw Project ........................................... 513
Setting up DragAndDrawActivity .................................................. 514
Setting up DragAndDrawFragment ............................................... 515
Creating a Custom View ............................................................... 516
Creating BoxDrawingView ............................................................ 516
Handling Touch Events ................................................................. 518
  Tracking across motion events ................................................ 519
Rendering Inside onDraw(...) ................................................... 522
Challenge: Rotations ................................................................. 524
33. Tracking the Device’s Location ............................................. 525
Getting Started with RunTracker ............................................. 525
  Setting up RunActivity ........................................................... 526
  Setting up RunFragment ........................................................ 527
Locations and the LocationManager ........................................ 528
Receiving Broadcast Location Updates ................................. 531
Updating the UI with Location Data ....................................... 532
Faster Answers: the Last Known Location ............................ 536
Testing Locations on Real and Virtual Devices ...................... 537
34. Local Databases with SQLite .................................................. 541
  Storing Runs and Locations in a Database .......................... 541
  Querying a List of Runs From the Database ....................... 548
  Displaying a List of Runs Using CursorAdapter ............... 550
  Creating New Runs ............................................................... 553
  Working with Existing Runs ................................................. 555
  Challenge: Identifying the Current Run .............................. 560
35. Loading Asynchronous Data With Loaders ......................... 561
  Loaders and the LoaderManager .......................................... 561
  Using Loaders in RunTracker .............................................. 563
  Loading the List of Runs ....................................................... 563
  Loading a Single Run .......................................................... 567
  Loading the Last Location for a Run ................................. 570
36. Using Maps ........................................................................ 573
  Adding the Maps API to RunTracker .................................... 573
    Use a real device to test maps ........................................... 573
    Install and use the Google Play services SDK ................ 573
    Obtain a Google Maps API key ........................................ 574
    Update RunTracker’s manifest ....................................... 574
  Showing the User’s Location on a Map ............................. 575
  Displaying a Run’s Path ....................................................... 579
  Adding Markers for Run Start and Finish ......................... 582
  Challenge: Live Updates ..................................................... 583
37. Afterword ........................................................................ 585
  The Final Challenge ............................................................. 585
  Shameless Plugs ................................................................. 585
  Thank You ......................................................................... 585
Index .................................................................................. 587
This page intentionally left blank
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 it 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 okay. 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.

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 forums.bignerdranch.com.
- Find someone who knows Android to help you out.

**How This Book Is Organized**

In 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 thirteen chapters. All are designed to teach you important concepts and techniques and give you direct experience using them.

<table>
<thead>
<tr>
<th>App</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoQuiz</td>
<td>In your first app, you will explore the fundamentals of Android projects, activities, layouts, and explicit intents.</td>
</tr>
<tr>
<td>CriminalIntent</td>
<td>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.</td>
</tr>
<tr>
<td>HelloMoon</td>
<td>In this small shrine to the Apollo program, you will learn more about fragments, media playback, resources, and localization.</td>
</tr>
<tr>
<td>NerdLauncher</td>
<td>Building this custom launcher will give you insight into the intent system and tasks.</td>
</tr>
<tr>
<td>RemoteControl</td>
<td>In this toy app, you will learn to use styles, state list drawables, and other tools to create attractive user interfaces.</td>
</tr>
<tr>
<td>PhotoGallery</td>
<td>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.</td>
</tr>
<tr>
<td>DragAndDraw</td>
<td>In this simple drawing app, you will learn about handling touch events and creating custom views.</td>
</tr>
<tr>
<td>RunTracker</td>
<td>This app lets you track and display on a map your travels around town (or around the world). In it, you will learn how to use location services, SQLite databases, loaders, and maps.</td>
</tr>
</tbody>
</table>

**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 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 three 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. It puts the listener’s method implementations right where you want to see them. In high performance contexts, 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.*

This is something we feel strongly about. Many Android developers still write activity-based code. We would like to challenge that practice. 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.

*We write apps to be compatible with Gingerbread and Froyo devices.*

The Android platform has changed with the introduction of Ice Cream Sandwich and Jelly Bean and soon Key Lime Pie. However, the truth is that half of devices in use still run Froyo or Gingerbread. (You will learn about the different and deliciously-named Android versions in Chapter 6.)

Therefore, we intentionally take you through the difficulties involved in writing apps that are backwards-compatible with Froyo or at least Gingerbread. It is easier to learn, teach, and program in Android if you start with the latest platform. But we want you to be able to develop in the real world where Gingerbread phones still make up more than 40% of devices.

**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 will be 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)`.

```java
@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 2.2 (Froyo) - Android 4.2 (Jelly Bean). As Android releases new versions, we will keep track of changes at forums.bignerdranch.com and offer notes on using this book with the latest version.

### The Necessary Tools

To get started, you will need the ADT (Android Developer Tools) Bundle. This includes:

**Eclipse**

an integrated development environment used for Android development. Because Eclipse is also written in Java, you can install it on a PC, a Mac, or a Linux computer. The Eclipse user interface follows the “native look-and-feel” of your machine, so your screen may not look exactly like screenshots in this book.

**Android Developer Tools**

a plug-in for Eclipse. This book uses ADT (Android Developer Tools) 21.1. You should make sure you have that version or higher.

**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

### Downloading and installing the ADT Bundle

The ADT Bundle is available from Android’s developer site as a single zip file.

1. Download the bundle from
2. Extract the zip file to where you want Eclipse and the other tools installed.

3. In the extracted files, find and open the eclipse directory and launch Eclipse.

If you are running on Windows, and Eclipse will not start, you may need to install the Java Development Kit (JDK6), which you can download from www.oracle.com.

If you are still having problems, return to http://developer.android.com/sdk/index.html for more information.

**Downloading earlier SDK versions**

The ADT Bundle provides the SDK and the emulator system image from the latest platform. However, you will need other platforms to test your apps on earlier versions of Android.

You can get components for each platform using the Android SDK Manager. In Eclipse, select Window → Android SDK Manager.

**Figure 1  Android SDK Manager**

For every version going back to Android 2.2 (Froyo), we recommend selecting and installing:

- the SDK Platform
- an emulator system image
- the Google APIs

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.

**A hardware device**

The emulator is useful for testing apps. However, it is good to have an actual Android device to run apps on as well. The last app in the book will require a hardware device.
Every instance of `Activity` has a lifecycle. During this lifecycle, an activity transitions between three possible 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:

```java
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)`**

```java
public class QuizActivity extends Activity {
    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)

```java
public class QuizActivity extends Activity {

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

    @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");
    }

} // End of onCreate(Bundle)
```

This code may cause an error regarding the `Log` class. If so, press Command+Shift+O (Ctrl+Shift +O) to organize your imports. Eclipse will then ask you to choose which class to import. Choose `android.util.Log`.

Now override five more methods in `QuizActivity`:

Listing 3.3 Overriding more lifecycle methods (QuizActivity.java)

```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:

```java
public class QuizActivity extends Activity {
    @Override
    public void onCreat(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.

To get to LogCat, select Window → Show View → Other... In the Android folder, find and select LogCat and click OK (Figure 3.2).

![Figure 3.2 Finding LogCat](image)

LogCat will open in the right half of your screen and annoyingly shrink your editor. It would be better if LogCat were at the bottom of the workbench window.

To get it there, drag from the tab of the LogCat view to the toolbar at the bottom-right corner of the workbench.
Using LogCat

Figure 3.3 Drag from LogCat tab to bottom-right toolbar

The LogCat view will close, and its icon (a horizontal Andy with Nyan rainbow feet) will appear in the toolbar. Click the icon, and LogCat will reopen at the bottom of the window.

Your Eclipse workbench should now look something like Figure 3.4. You can resize the panes in LogCat by dragging their boundaries. This is true for any of the panes in the Eclipse workbench.
Run GeoQuiz, and messages will start appearing fast and furiously in LogCat. Most of the messages will be system output. Scroll to the bottom of the log to find your messages. In LogCat’s Tag column, you will see the TAG constant you created for **QuizActivity**.

(If you do not see any messages in LogCat, LogCat may be monitoring the wrong device. Select Window → Show View → Other... and open the Devices view. Select the device you are currently running on and then return to LogCat.)

To make your messages easier to find, you can filter the output using the TAG constant. In LogCat, click the green + button at the top of the lefthand pane to create a message filter. Name the filter **QuizActivity** and enter **QuizActivity** in the by Log Tag: field (Figure 3.5).
Click OK, and a new tab will open showing only messages tagged QuizActivity (Figure 3.6). Three lifecycle methods were called after GeoQuiz was launched and the initial instance of QuizActivity was created.

(If you are not seeing the filtered list, select the QuizActivity filter from LogCat’s lefthand pane.)

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().

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 3.8 Pressing the Home button stops the activity**

On the device, pull up the task manager. On newer devices, press the Recents button next to the Home button (Figure 3.9). On devices without a Recents button, long-press the Home button.

**Figure 3.9 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.
Finally, imagine a small pop-up window that only partially covers the activity. When one of these appears, the activity behind it is paused and cannot be interacted with. The activity will be resumed 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. (For the emulator, press Control+F12/Ctrl+F12 to rotate.)

After rotating, GeoQuiz will display the first question again. Check LogCat to see what has happened.

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

First, minimize LogCat. (If you accidentally close LogCat instead, you can always re-open it from Window → Show View...)

Next, in the package explorer, right-click the res directory and create a new folder. Name this folder layout-land.

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.

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/
Device configurations and alternative resources

You will also get more practice working with configuration qualifiers in Chapter 15.

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/`.

Let’s make some changes to the landscape layout so that it is different from the default. Figure 3.12 shows the changes that you are going to make.

**Figure 3.12** 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` 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 3.12. You can use Listing 3.4 to check your work.
Listing 3.4 Tweaking the landscape layout (layout-land/activity_quiz.xml)

```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:drawableRight="@drawable/arrow_right"
            android:drawablePadding="4dp"
            android:paddingHorizontal="8dp"/>
    </FrameLayout>
</LinearLayout>

Run GeoQuiz again. Rotate the device to landscape to see the new layout. Of course, this is not just a new layout; it is a new QuizActivity as well.
Saving Data Across Rotation

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 recreating 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(...) asks 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)`

```java
@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 recreate 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)**

```java
public class QuizActivity extends Activity {
    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";
    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)**

```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)

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

Run GeoQuiz and press Next. No matter how many device or user 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 objects that implement the `Serializable` interface. When you are creating custom classes, be sure to implement `Serializable` if you want to save them in `onSaveInstanceState(...)`.

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 3.14).
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 data 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. This app is included with most system images used on the emulator.

Figure 3.15 Finding the Settings app

Launch Settings and select Development options. Here you will see many possible settings. Turn on the setting labeled Don't keep activities.
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.

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 3.17. Each level has a corresponding method in the `Log` class. Sending output to the log is as simple as calling the corresponding `Log` method.
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 = mQuestionBank[mCurrentIndex];
try {
    } 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);
    }
```
This page intentionally left blank
This page intentionally left blank
Index

Symbols
9-patch images, 409, 410
@+id, 18, 187
@Override, 56
@SuppressLint("HandlerLeak"), 444
@SuppressWarnings("deprecation"), 306
@SuppressWarnings("unchecked"), 445
@TargetApi annotation, 119

A
aapt (Android Asset Packing tool), 27
AbsListView.MultiChoiceModeListener interface, 289
action bar
  action view in, 463
  android:showAsAction, 256
  compatibility and, 294
  contextual (see contextual action bar)
  features, 253
  hiding, 315
  options menu, 253
  overflow menu, 256
ActionBar
  onCreateOptionsMenu(…), 257
  setDisplayHomeAsUpEnabled(…), 263
ActionBarSherlock library, 294–298
ActionMode, 289, 293
ActionMode.Callback interface, 289-291, 293
activities
  (see also Activity, fragments)
abstract fragment-hosting activity, 172
adding to project, 89-112
as controller, 36
back stack of, 110-112, 380
base, 380
child, 90, 104
defined, 2
fragment transactions and, 365
handling configuration changes in, 510
hosting fragments, 128, 134-137, 232, 233
label (display name), 377
launcher, 109, 317
lifecycle and fragments, 146
lifecycle of, 53, 61, 67, 68
managing fragments, 365-373
metadata for, 265
overriding methods, 54
passing data between, 101-109
record, 67
rotation and, 61-65
running from command line, 317
searchable, 455, 458
starting from fragment, 191
starting in current task, 381
starting in new task, 382
states of, 53, 67
tasks and, 380
UI flexibility and, 126
Activity
  as Context subclass, 23
FragmentActivity, 131
getIntent(), 103, 193
invalidateOptionsMenu(), 477
lifecycle methods, 53-61
onActivityResult(…), 106
onCreate(…), 16, 53, 55
onDestroy(), 53
onPause(), 53
onResume(), 53, 197
onSaveInstanceState(…), 65-67, 241, 243
onSearchRequested(), 453
onStart(), 53
onStop(), 53
setContentView(…), 16
setResult(…), 106
SingleFragmentActivity, 172, 174-176, 361
startActivity(…), 99
startActivityForResult(…), 104
startSearch(…), 465
activity record, 67
ActivityInfo, 379
ActivityManager
  back stack, 110, 112
  starting activities, 99, 101, 102, 106, 107
ActivityNotFoundException, 101
Adapter interface, 179
adapters
  adapter views and, 179
  creating custom, 187
  defined, 179
  implementing, 180
creating, 24
for tablets, 359

B
Back button, 59, 60, 380
back stack, 110
background threads
dedicated, 438
updating UI from, 430
using AsyncTask for, 419, 422
beginTransaction() (FragmentTransaction), 143
Bitmap.recycle(), 337
breakpoints
(see also debugging)
exception, 83, 84
removing, 83
setting, 80-83
broadcast intents
defined, 485
ordered, 495-499
permissions and, 493-495
registered in code, 490-492
sending, 489, 490
broadcast receivers
defined, 485, 486
dynamic, 490-492
implementing, 486, 487
intent filters and, 487
location updates and, 546-548
long-running tasks and, 499
permissions and, 492-495
registering, 486
uses for, 488, 489
build errors
(see also debugging)
debugging, 87
missing imports, 19
build process, 26
build SDK version, 116
build target, 116
Build.VERSION.SDK_INT, 119
Bundle
for fragment arguments, 195, 196
in onCreate(...), 66
in onSaveInstanceState(...), 65
putCharSequence(...);, 195
putInt(...);, 195
putSerializable(...), 195
Button
adding icon to, 48
example, 9
v. ImageButton, 50
inheritance, 50
button
adding ID, 18
buttons, 50
C
caching, 450
Calendar, 219
Callbacks interface, 365-373
camera
accessing, 304-306
checking for, 306, 314, 315
compatibility and, 305
different types of, 305, 314
as exclusive-access resource, 300, 304
image file sizes and, 299
image handling and, 334-338
implicit intent for, 299
parameters of, 310, 324
permissions for, 303
setting picture size, 324
setting preview size, 310
as Surface client, 307-309
taking pictures, 321-323
Camera
getSupportedPictureSizes(), 324
getSupportedPreviewSizes(), 310
open(...), 304
Parameters inner class, 310, 324
PictureCallback interface, 321-323
release(), 304
setPreviewDisplay(...), 308
ShutterCallback interface, 321-323
startPreview(), 308
stopPreview(), 308
takePicture(...), 321, 323
cancel(...) (AlarmManager), 474, 475
cancel(...) (AsyncTask), 432
Canvas, 522
CheckBox, 151
focusable, 189
CHOICE_MODE_MULTIPLE_MODAL, 288
choosers, creating, 352
codemnames, version, 113
commands (IntentService), 467
compatibility
  action bar and, 294
  with ActionBarSherlock library, 294-298
  ActionBarSherlock library, 296
  Android Lint and, 118-120
  camera and, 305
  context actions and, 283
  for ListFragment, 170
  fragments and, 131, 132
  importance of, xix, 113-115
  issues, 114
  layout attributes and, 256
  minimum SDK version and, 116
  options menus and, 283
  strategies, 293, 294
  with support library, 131, 132
  using conditional code for, 118
  wrapping code for, 118-120
ComponentName, 380
components, 100
configuration changes, 61, 65, 238
configuration qualifiers
  defined, 62
  for language, 245
  listed, 247
  multiple, 249, 250
  precedence of, 247-250
  for screen density, 46, 246
  for screen orientation, 62
  for screen size, 364, 373, 374
ConnectivityManager, 470
contacts
  getting data from, 355
  permissions for, 356
  container views, 136, 144
  content assist, using, 23, 24
ContentProvider, 355
ContentResolver, 355
Context
  application-wide, 169
  explicit intents and, 100
  getSharedPreferences(…), 461
  for I/O, 274, 281
  resource IDs and, 23
  singletons and, 169
context menus
  compatibility and, 283
  defining menu, 284
  determining selected item, 287
  example of, 283
  implementing, 284-287
ContextMenu.ContextMenuInfo interface, 286
contextual action bar, 283, 287
  (see also contextual action mode)
contextual action mode
  callbacks, 289
  defined, 287
  defining menu, 284
  enabling multiple selection, 288
  implementing in ListView, 288-291
  implementing in other views, 293
contextual actions, 253, 283
controller objects, 36
conventions
  class naming, 6
  extra naming, 102
  package naming, 3
  variable naming, 19, 33
corner (drawable), 401
create() (AlertDialog.Builder), 214
create(…) (MediaPlayer), 234
createChooser(…) (Intent), 352
Cursor, 541, 548
CursorAdapter, 551
CursorLoader, 561
CursorWrapper, 549

D
Dalvik Debug Monitor Service (see DDMS)
Dalvik virtual machine, 384
/data/app directory, 281
/data/data directory, 273
databases, 541-554
Date, 219
DatePicker, 215
DatePickerDialog, 211
DDMS (Dalvik Debug Monitor Service), 75
DDMS perspective, 75
Debug view, 80
debugging
  (see also Android Lint)
build errors, 87

- crash, 76
- crash on unconnected device, 77
- misbehaviors, 77
- online help for, 88

R, 87
- running app with debugger, 80
- stopping debugger, 82

DEFAULT (Intent), 383
default resources, 246
delayed execution, 472, 473
density-independent pixel (see dip)
derecprecation
- in Android, 342
- using deprecated methods, 306, 342
detach(…) (FragmentTransaction), 208

Dev Tools, 70
developer documentation, 120-122
devices
- configuration changes and, 61
- configuring language settings, 245
- hardware, 24
- virtual, 24, 359
 Devices view, 44, 75

Dialog, 211
DialogFragment, 211
- onCreateDialog(…), 213
- show(…), 214
- showing image in, 338
dialogs, 211-217
dip (density-independent pixel), 155
documentation, 120-122
doInBackground(…) (AsyncTask), 419
dp (density-independent pixel), 155
draw() (View), 522
draw9patch tool, 410
drawables, 399, 401
- 9-patch images, 409
- corner, 401
documentation, 401
- gradient, 401, 405
- inset, 404, 405
- layer-list, 404, 405
- referencing, 48
- state list, 291, 402, 403
- stretchable, 406
drawing
Canvas, 522

in onDraw(…), 522
Paint, 522
to Surface, 306-309

E

Eclipse
- auto-complete, 23, 24
- build process, 26
- code style preferences, 33
- content assist, 23, 24
- creating new classes, 32
- debugger, 80, 81
- (see also debugging)
editor, 7
- generating getter and setter methods, 33, 34, 133
- graphical layout tool, 157, 251
- installing, xx
- organizing imports, 19, 24
- package explorer, 7
- panes, resizing, 57
- preferences, 33
- res/values directory, 14
- src directory, 15
- tab group, 7
- views, 75
  - Debug view, 80
  - Devices view, 44, 75
  - Lint Warnings view, 86
  - moving, 7, 56
  - Variables view, 81
- workbench window, 7

EditText, 137
emulator
- (see also virtual devices)
- rotating, 48
- running on, 24
- system images for, xx
- for tablets, 359
errors
- (see also debugging)
- missing imports, 19
- escape sequence (in string), 39
- exception breakpoints, 83, 84
- exceptions, 76
- explicit intents
  - creating, 100
creating at runtime, 380
purpose, 101
external storage
checking for, 280
defined, 273
using, 280
extras
defined, 102
fragments retrieving from activity, 193
as key-value pairs, 102
naming, 102
putting, 102, 103
retrieving, 103
structure of, 102

F
file explorer, 84
file I/O, 273-280
fill_parent, 13
finalizer execution, 337
Flickr API, 422
floating context menus (see context menus)
fluent interface, 143
format string, 348
Fragment
from native libraries, 147
from support library, 131
getActivity(), 171, 191, 193
getArguments(...), 196
getTargetFragment(), 221
getTargetRequestCode(), 221
newInstance(...), 195
onActivityResult(...), 217
onContextItemSelected(...), 284, 286
onCreate(Bundle), 140
onCreateContextMenu(...), 284
onCreateOptionsMenu(...), 257, 258
onCreateView(...), 140
onPrepareOptionsMenu(...), 477
onSaveInstanceState(...), 140, 237
registerForContextMenu(...), 285
setArguments(...), 195
setHasOptionsMenu(...), 258
setRetainInstance(...), 237
setTargetFragment(...), 221
startActivityForResult(...), 199
fragment arguments, 193, 195, 196
fragment transactions, 143-145, 365, 367, 368
(see also FragmentTransaction)
FragmentActivity (from support library), 131
FragmentManager
adding fragments to, 142-145
fragment lifecycle and, 145-147
layout fragments and, 232
onResume(), 197
responsibilities, 142
role in rotation, 237
FragmentPagerAdapter, 208
fragments
(see also fragment transactions,
FragmentManager)
accessing extra in activity’s intent, 193
v. activities, 126
activity lifecycle and, 146
adding in code, 136
adding to FragmentManager, 142-147
arguments of, 195, 196
as composable units, 126, 365
Callbacks interface, 365-373
compatibility and, 131, 132
container views for, 136, 361
creating, 137
creating from support library, 139
defined, 126
delegating functionality to activity, 365
handling activity functions, 171
hosting, 128, 134-137, 232, 233
hosting options, 135
implementing lifecycle methods, 139
inflating layouts for, 140
layout, 135, 232, 233
lifecycle diagram, 146
lifecycle methods, 146
lifecycle of, 134, 135, 145-147, 233
maintaining independence of, 195, 365
passing data between (same activity), 217
passing data with fragment arguments, 218
reasons for, 126, 127, 147
retaining, 237-243
rotation and, 238-240, 243
setting a result from, 326-328
setting listeners in, 141
starting activities from, 191
support library and, 131, 132

592
without support library, 147
UI flexibility and, 126
widgets and, 141
**FragmentStatePagerAdapter**, 204
getCount(), 204, 205
getItem(...), 204, 205
setOffscreenPageLimit(...), 206
**FragmentTransaction**
add(...), 144
beginTransaction(), 143
detach(...), 208
remove(...), 208
**FrameLayout**
as container view for fragments, 136, 361
described, 63
replacing with merge, 87

**G**

**Gallery**, 209
gen directory, 16
getAction() (MotionEvent), 518
getActiveNetworkInfo()
(ConnectivityManager), 470
getActivity() (Fragment), 171, 191, 193
getArguments(...) (Fragment), 196
getBackgroundDataSetting()
(ConnectivityManager), 470
getBooleanExtra(...) (Intent), 103
getBroadcast(...) (PendingIntent), 530
getCount() (FragmentStatePagerAdapter), 204, 205
defaultSharedPreferences(...)
(PreferenceManager), 461
getInputStream() (HttpURLConnection), 418
getIntent() (Activity), 103, 193
getItem(...) (FragmentStatePagerAdapter), 204, 205
getListAdapter() (ListFragment), 184
getLoaderManager() (LoaderManager), 562
getMenuInfo() (MenuInfo), 286
getOutputStream() (HttpURLConnection), 418
getReadableDatabase() (SQLiteOpenHelper), 543
getSearchableInfo(...) (SearchView), 464
getService(...) (PendingIntent), 473
getSharedPreferences(...) (Context), 461
getTargetFragment() (Fragment), 221

**getTargetRequestCode()** (Fragment), 221
getter and setter methods, generating, 33, 34, 133
**getWritableDatabase()** (SQLiteOpenHelper), 543
Google APIs, 117, 526
Google Maps API, 573, 574
Google Play services SDK, 573
GPS
adding ACCESS_FINE_LOCATION permission, 531
adding ACCESS_MOCK_LOCATION permission, 537
adding uses-feature, 531
gradient (drawable), 401, 405
graphical layout tool, 157-165, 251

**GridView**
(see also AdapterView, ListView)
for display columns, 416
setOnItemClickListener(...), 503

**H**

**Handler**, 441, 445
handlers, 441-449
**HandlerThread**
vs. **AsyncTask**, 449
handling downloads, 439
hardware devices, 24
hardware menu key, 256
-hdpi suffix, 46
hierarchical navigation, 262
**HOME** (Intent), 383
Home button, 60
home screen, 383
Honeycomb, 114
HTTP networking, 414, 417-419, 422

**HttpURLConnection**
class, 418
getInputStream(), 418
getOutputStream(), 418

**I**
icons, 256, 257
image handling, 334-338
**ImageButton**, 50
**ImageView**, 331
implicit intents
action, 350, 376
ACTION_CALL category, 357
ACTION_DIAL category, 357
ACTION_PICK category, 353
ACTION_SEND category, 351
benefits of using, 345
for camera, 299
categories, 350, 376
CATEGORY_DEFAULT, 377
data, 350
v. explicit intent, 350
parts of, 350
sending with AlarmManager, 473
type, 350
include (layout), 394
inflating layouts, 16, 140
initLoader(...) (LoaderManager), 562
InputStream
for delivering bytes, 418
read(), 418
inset (drawable), 404, 405
Intent
addFlags(...), 382
constructors, 100
createChooser(...), 352
getBooleanExtra(...), 103
putExtra(...), 102, 193
setClassName(...), 380
setComponent(...), 380
intent filters
BOOT_COMPLETED, 486
explained, 350
MAIN, 109
SEARCH, 457
SHOW_NOTIFICATION, 491
intent services
processing commands, 468
purpose, 467
intents
(see also broadcast intents, explicit intents, extras, implicit intents, Intent)
communicating with, 100, 101
implicit v. explicit, 345
IntentService, 467
internal storage, 273
invalidate() (View), 521
invalidateOptionsMenu() (Activity), 477
is prefix for variable names, 33
JavaScript Object Notation (see JSON)
JavaScript, enabling, 507
Javascript, injecting objects, 510
JSON (JavaScript Object Notation)
defined, 274
loading, 278-280
saving app data as, 274-277
serialization, 276
v. SQLite, 541
toJSON(), 275
JSONArray, 275
JSONObject, 276
-land qualifier, 62
language settings, 245
launch mode
for search, 456
types, 459
launcher activities, 109
LAUNCHER category, 109, 377
layer-list (drawable), 404, 405
layout attributes
android:contentDescription, 51
android:drawable, 292
android:drawablePadding, 48
android:drawableRight, 48
android:focusable, 189
android:icon, 256
android:layout_gravity, 63
android:layout_height, 13
android:layout_width, 13, 301
android:layout_weight, 13, 301
android:orientation, 13
android:padding, 156
android:stretchColumns, 390
android:stretchMode, 416
compatibility and, 256
layout fragments, 232, 233
layout parameters (layout_), 156
LayoutInflate, 27, 141
layouts
(see also graphical layout tool, layout attributes, widgets)
alternative, 62-64
creating new, 91
defined, 2
defining in XML, 11-13
design documentation, 155
include, 394
inflating, 16, 140
landscape, 62-64
managing multiple, 165
merge, 397
naming, 6, 92
previewing, 14, 94, 251
root element, 12
testing, 251
view hierarchy and, 12
layout_weight attribute
  Android attribute, 164
-ldpi suffix, 46
library projects, 294
LinearLayout, 9, 12
lint (see Android Lint)
Lint Warnings view, 86
list items
  activated state, 291
  changing background, 291
  customizing display of, 184
  handling selection, 183, 288, 291
  setting text of, 182
list-detail interfaces, 125, 201, 359-369
listeners
  defined, 20
  as interfaces, 20
  setting in fragments, 141
  setting up, 20-24
ListFragment
  compatibility and, 170
  creating, 170, 171
  defined, 168
  getListAdapter(), 184
ListView and, 171
  onListItemClick(...), 183
  setListAdapter(...), 181
  support library class, 170
lists
  (see also adapters, ListFragment, ListView)
  basic mechanics of, 178-184
  displaying, 167
  getting item data, 179
  getting the adapter, 184
  setting adapter, 181
ListView
  (see also AdapterView)
  basic operations of, 178
  default implementation, 171
  enabling multiple selection, 288
  setChoiceMode(...), 288
  setEmptyView(...), 270
Loader, 561
LoaderCallbacks, 562
  onCreateLoader(...), 566
  onLoaderReset(...), 566
  onLoadFinished(...), 566
LoaderManager, 562
loaders, 561-571
loadLabel(...) (ResolveInfo), 378
local files (see app data)
localization, 245, 246
location updates
  (see also GPS)
  delivery mechanisms, 529
  last known location, 536
  receiving broadcast updates, 531
  testing, 533
  testing real device with a TestProvider, 537
  testing virtual device with DDMS, 537
LocationListener, 529
LocationManager, 528-530
Log, 54
Log.d(...), 78
LogCat, 56-61
  (see also logging)
logging
  levels, 70
  Log.d(...), 78
  messages, 54
  methods, 70
  of exceptions, 78
  of stack traces, 78
  TAG constant, 54
long-press, simulating on emulator, 287
Looper, 439, 442
LRU (least recently used) caching strategy, 450
LRUCache, 450

M
m prefix for variable names, 19
MAIN category, 109, 377
Index

main thread, 421

makeText(...) (Toast), 23

manifest
(see also manifest attributes)
adding camera permissions to, 303
adding GPS permissions to, 531
adding network permissions to, 419
adding service to, 468
adding uses-feature GPS to, 531
adding uses-permission INTERNET, 419
Android versions in, 115
build process and, 26
declaring Activity in, 97
defined, 97
uses-feature, 304
uses-permission, 303
uses-sdk, 115

manifest attributes
android:configChanges, 510
android:protectionLevel, 495
android:resource, 457
android:screenOrientation, 304
android:theme, 231

maps
adding live updates to, 583
adding markers to, 582
displaying, 575
displaying a path on, 579
master-detail interfaces, 125, 201, 359-369
match_parent, 13
-mdpi suffix, 46

MediaPlayer, 227
create(...), 234
documentation, 236
on different thread, 235
release(), 234
stop(), 234
video playback, 236

MenuItem, 261, 286
menus
(see also action bar, context menus, options menus)
android:actionViewClass, 463
creating XML file for, 255
defining in XML, 255
as resources, 255
search and, 463
merge (layout), 87, 397

Message, 441
message handlers, 441-449
message loop, 438
message queue, 438
messages, 441-449
meta-data
parent activity, 265
searchable, 457
minimum required SDK, 115
minSdkVersion, 115, 116
model layer, 35
model objects, 35
motion events, handling, 518-521

MotionEvent
actions, 518
class, 518
getAction(), 518

MVC (Model-View-Controller), 35-37

N
namespace, Android XML, 12
navigateUpFromSameTask(...) (NavUtils), 265

navigation
types of, 262

NavUtils
for ancestral navigation, 265
navigateUpFromSameTask(...), 265
network, checking availability of, 470
networking (HTTP), 414, 417-419, 422

NetworkOnMainThreadException, 421
newInstance(...)
9-patch images, 409, 410

Notification, 478
NotificationManager, 478
notifications, 478, 479
notify(...)

NullPointerException, 76

O
onActionItemClicked(...)
(ActionMode.Callback), 289
onActivityResult(...) (Activity), 106
onActivityResult(...) (Fragment), 217

OnCheckedChangeListener interface, 153
onClick(View) (OnClickListener), 21

OnClickListener interface, 21
onContextItemSelected(...) (Fragment), 284, 286
onCreate(Bundle) (Activity), 16, 53-55
onCreate(...) (Fragment), 140
onCreate(...) (SQLiteOpenHelper), 543
onCreateActionMode(...) (ActionMode.Callback), 289
onCreateContextMenu(...) (Fragment), 284
onCreateOptionsMenu(...) (Action), 257
onCreateOptionsMenu(...) (Fragment), 257, 258
onCreateView(...) (Fragment), 140
onDestroy() (Activity), 53
onDestroyActionMode(...) (ActionMode.Callback), 289
onDraw(...) (View), 522
onHandleIntent(...) (IntentService), 467
onItemCheckedStateChanged(...) (MultiChoiceModelListener), 289
onListItemClickListener(...) (ListAdapter), 183
onLoaderReset(...) (LoaderCallbacks), 566
onLocationChanged(...) (LocationListener), 529
onOptionsItemSelected(...) (Fragment), 261
onPageChangeListener interface (ViewPager), 207
onPageScrolled(...) (ViewPager), 208
onPageScrollStateChanged(...) (ViewPager), 208
onPageSelected(...) (ViewPager), 208
onPause() (Activity), 53
onPostExecute(...) (AsyncTask), 430
onPrepareActionMode(...) (ActionMode.Callback), 289
onPrepareOptionsMenu(...) (Fragment), 477
onProgressChanged(...) (WebChromeClient), 509
onProgressUpdate(...) (AsyncTask), 432
onReceivedTitle(...) (WebChromeClient), 509
onRestoreInstanceState(...) (View), 524
onResume() (Activity), 53, 197
onResume() (FragmentManager), 197
onSaveInstanceState(...) (FragmentManager), 241, 243
onSaveInstanceState(...) (Activity), 65-67, 69, 70
onSaveInstanceState(...) (Fragment), 140, 237
onSaveInstanceState(...) (View), 524
onSearchRequested() (Activity), 453
onStart() (Activity), 53
onStop() (Activity), 53
onTextChanged(...) (TextWatcher), 141
onTouchEvent(...) (View), 518
onTouchInteraction(...) (View), 518
onUpgrade(...) (SQLiteOpenHelper), 543
open(...) (Camera), 304
openConnection(...) (URL), 418
openFileOutput(...) (Reader), 279
options menus
  accessing on pre-Honeycomb, 259
  android:showAsAction, 256
  compatibility and, 283
  creating, 257
  defined, 253
  determining selected item, 261
  populating with items, 257
  pre/post-Honeycomb behavior, 260
  responding to selections, 260
  updating dynamically, 477
  updating dynamically (post Honeycomb), 477
organizing imports, 19, 24
OutputStream, 276
OutputStreamWriter, 276
overflow menu, 256
@Override, 56

P
PackageManager class, 357
metadata and, 265
queryIntentActivities(...), 357
packages, naming, 3
padding, 156
Paint, 522
Parcelable, 524
parse XML, 426
PendingIntent details, 474
getBroadcast(...), 530
ggetService(...), 473
location updates and, 529
persistent data
  with JSON, 273
  using shared preferences, 461
perspectives
Index

DDMS perspective, 75
Debug perspective, 80
defined, 75
Java perspective, 75
resetting, 75
switching between, 75
PointF, 519
post(...)(Handler), 448
PreferenceManager
getDefaultSharedPreferences(...), 461
preferences (Eclipse), 33
preloading, 450
processes, 384, 385
progress indicator
hiding, 509
spinner type, 320
updating from background thread, 432
ProgressBar, 320
projects
adding resources, 46
configure, 3
creating, 3-6
gen directory, 16
layout, 6
library, 294
res/layout directory, 16
res/menu directory, 255
res/raw directory, 228
res/values directory, 16
setting package name, 3
setting project name, 3
src directory, 15
protection level values, 495
publishProgress(...)(AsyncTask), 432
putCharSequence(...); (Bundle), 195
putExtra(...)(Intent), 193
putInt(...); (Bundle), 195
putSerializable(...)(Bundle), 195
Q
queryIntentActivities(...)(PackageManager), 357
R
R, 16
randomUUID(), 133
read()(InputStream), 418
Recents button, 60
cycle() (Bitmap), 337
registerForContextMenu(...)(Fragment), 285
RelativeLayout, 185, 186
release key, 26
release()(Camera), 304
release()(MediaPlayer), 234
remove(...)(FragmentTransaction), 208
request code (Activity), 104
requestLocationUpdates(...)(LocationManager), 530
res/layout directory, 16
res/menu directory, 255
res/raw directory, 228
res/values directory, 14, 16
ResolveInfo, 377, 378
resource IDs, 16-18
+ prefix in, 18, 187
  multiple layouts and, 165
  syntax, 187
resources
(see also configuration qualifiers, drawables,
layouts, menus, strings)
adding, 46
alias, 363-365
alternative, 247-250
default, 246
defined, 16
directory structure, 251
localizing, 245, 246
location of, 16
naming requirements, 251
referencing in XML, 47
string, 13, 14
styles, 392, 393
restartLoader(...)(LoaderManager), 562
result code (Activity), 105
retained fragments, 237-243
retainInstance property (Fragment), 237, 238
rotation
  activity lifecycle and, 61-65
  before fragments, 243
  onSaveInstanceState(...) and, 241, 243
  preventing, 304
  saving data across, 65-67
  running on device, 44, 45
s prefix for variable names, 33
scale-independent pixel (see sp)
screen orientation, 62
forcing, 304
screen pixel density, 46, 154, 246
screen size, determining, 373, 374
ScrollView, 347
SD card, 273
SDK versions
build target, 116
codenames, 113
installing, xxi
listed, 113
minimum required, 115
target, 115
updating, xxi
search
configuration, 456
configuring emulator for, 458
coordinating with SearchManager service, 456
hardware button, 453
how it works, 458
integrating into app, 451
intent filter, 457
metadata, 457
post Honeycomb, 463
program-initiated, 453
specifying details of, 465
searchable activity, 458
searchable.xml, 455
SearchView
bug, 465
class, 463
getSearchableInfo(…), 464
selector (XML element), 292
services
adding to manifest, 468
bound, 481
lifecycle of, 480
locally bound, 482
non-sticky, 481
notifying user, 478
purpose of, 467
remotely bound, 483
sticky, 481
setArguments(…)(Fragment), 195
setChoiceMode(…)(ListView), 288
setClassName(…)(Intent), 380
setComponent(…)(Intent), 380
setContentView(…)(Activity), 16
setDisplayHomeAsUpEnabled(…)(ActionBar), 263
setEmptyView(…)(AdapterView), 270
setHasOptionsMenu(…)(Fragment), 258
setJavaScriptEnabled(…)(WebView), 507
setListAdapter(…)(ListFragment), 181
setOnScreenPageLimit(…)
(FragmentStatePagerAdapter), 206
setOnClickListener(…), 21
setOnItemClickListener(…)(GridView), 503
setOnTouchListener(…)(View), 518
setPositiveButton(…)(AlertDialog.Builder), 213
setPreviewDisplay(…)(Camera), 308
setRepeating(…)(AlarmManager), 474
setResult(…)(Activity), 105, 106, 199
setRetainInstance(…)(Fragment), 237
setTargetFragment(…)(Fragment), 221
setter methods (see getter and setter methods)
setText(…)(TextView), 104
setTitle(…)(AlertDialog.Builder), 213
setType(…)(SurfaceHolder), 306
setView(…)(AlertDialog.Builder), 215
shadow effect (adding), 404
shape drawables, 401
shared preferences, 461, 462
SharedPreferences, 461
SharedPreferences.Editor, 462
shouldOverrideUrlLoading(…)
(WebViewClient), 507
show() (Toast), 23
show(…)(DialogFragment), 214
simulator (see emulator)
SingleFragmentActivity, 172, 174-176, 361
singletons, 169
solutions file, 45
sp (scale-independent pixel), 155
SQLite, 541-560
SQLiteDatabase, 541, 556
SQLiteDatabase query(…)(SQLiteDatabase), 556
SQLiteOpenHelper, 541, 543
src directory, 15
Index

stack traces
  in LogCat, 76
  logging of, 78
startActionMode(…) (Activity), 293
startActivity(…) (Activity), 99
startActivityForResult(…) (Activity), 104
startActivityForResult(…) (Fragment), 199
startPreview() (Camera), 308
startSearch(…) (Activity), 465
  state list (drawable), 402-404
  status bar, hiding, 315
stop() (MediaPlayer), 234
stopPreview() (Camera), 308
stretchable (drawable), 406
string resources
  creating, 14
  defined, 13
  referencing, 48
strings file
  default, 14
  defined, 13
strings.xml, 14
styles, 392, 393, 401, 404
  defined, 153
  inheritance, 398
  themes and, 154
styles.xml, 392
support library, 131, 132
SupportMapFragment, 575
@SuppressLint(“HandlerLeak”), 444
@SuppressWarnings(“deprecation”), 306
@SuppressWarnings(“unchecked”), 445
Surface
  client, 306
  defined, 306
  lifecycle of, 306
SurfaceHolder interface
  setType(…), 306
  using, 306-309
SurfaceHolder.Callback interface, 308
SurfaceView
  defined, 300
  getHolder(), 306
  SurfaceHolder and, 306-309
  -sw600dp suffix, 364
  system icons, 256, 257

T
TableLayout, 230
TableRow, 230
tables
  creating virtual devices for, 359
  user interfaces for, 359-369
TAG constant, 54
target fragment, 221
target SDK version, 115
@TargetApi annotation, 119
targetSdkVersion, 115, 116
task manager, 60, 380
tasks
  and Back button, 380
  defined, 380
  v. processes, 384, 385
temporal navigation, 262
TextView
  example, 9
  inheritance, 50
  setText(…), 104
TextWatcher interface, 141
themes, 154, 231
threads
  background (see background threads)
  main, 421
  message queue, 438
  processes and, 384
  as sequence of execution, 421
  UI, 421
Toast, 23
toasts, 22-24
touch events
  handling, 518-521
  intercepting, 321

U
UI fragments (see fragments)
UI thread, 421
Up button, 262-264
Uri.Builder
  appendQueryParameter(…), 424
  for escaping query strings, 424
URL
  for making URL from string, 418
  openConnection(), 418
URLConnection, 418
Index

V

variable names
  conventions for, 33
  prefixes for, 33

Variables
  view, 81

versions
  (Android SDK) (see SDK versions)
  (firmware), 113

video playback, 227, 236

VideoView, 236

View
  (see also views, widgets)
    draw(), 522
    invalidate(), 521
    OnClickListener interface, 20
    onDraw(...), 522
    onRestoreInstanceState(...), 524
    onSaveInstanceState(), 524
    onTouchEvent(...), 518
    setOnTouchListener(...), 518
  subclasses, 8, 50

  view layer, 35
  view objects, 35

ViewGroup, 12, 63

ViewPager, 201-210
  in support library, 204
  internals of, 209
    OnPageChangeListener, 207
    onPageScrolled(...), 208
    onPageScrollStateChanged(...), 208
    onPageSelected(...), 208

views
  (see also widgets, layouts)
  creating, 516
  custom, 516-518
  persisting, 524
  simple v. composite, 516

touch events and, 518-521

using fully-qualified name in layout, 517

virtual devices
  (see also emulator)
  creating, 24
  for tablets, 359
  testing low-memory handling, 69

W

web content
  browsing via implicit intent, 503
  displaying in browser, 501
  displaying in WebView, 501
  displaying within an activity, 503
  enabling JavaScript, 507
  web rendering events, responding to, 507

WebChromeClient
  for enhancing appearance of WebView, 508
  interface, 508
  onProgressChanged(...), 509
  onReceivedTitle(...), 509

WebSettings, 507

WebView
  for presenting web content, 504
  handling rotation, 509

WebViewClient, 507

widgets
  attributes of, 11, 156
  Button, 9, 50
  CheckBox, 151
  DatePicker, 215
  defined, 8
  defining in XML, 11-13
  EditText, 137
  FrameLayout, 63
  ImageButton, 50
  LinearLayout, 9, 12
  padding, 156
  references, 20
  ScrollView, 347
  TableLayout, 230
  TableRow, 230
  TextView, 9
  in view hierarchy, 12
  as view layer, 35
  wiring in fragments, 141
  wiring up, 19
Index

wrap_content, 13
Writer, 276

X
-xlarge suffix, 364
XML
   Android namespace, 12
       referencing resources in, 47
XML drawables (see drawables)
XML parsing, 426
XmlPullParser, 426