



Phil Dutson

Android™ Development Patterns

Best Practices for Professional Developers



FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Android™ Development Patterns

This page intentionally left blank

Android™ Development Patterns

Best Practices for Professional Developers

Phil Dutson

◆ Addison-Wesley

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

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

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

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

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

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2015958569

Copyright © 2016 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

Google Play is a trademark of Google, Inc.

Android is a trademark of Google, Inc.

ISBN-13: 978-0-133-92368-1

ISBN-10: 0-133-92368-1

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

First printing: February 2016

Editor-in-Chief

Mark Taub

Executive Editor

Laura Lewin

Development Editor

Sheri Replin

Managing Editor

Kristy Hart

Project Editor

Elaine Wiley

Copy Editor

Bart Reed

Indexer

Tim Wright

Proofreader

Laura Hernandez

Technical Reviewers

Romin Irani

Douglas Jones

Raymond Rischpater

Editorial Assistant

Olivia Basegio

Cover Designer

Chuti Prasertsith

Compositor

Nonie Ratcliff



*To all of those who believe in magic,
especially the digital kind.*



This page intentionally left blank

Contents

Preface xiv

1 Development Tools 1

- Android Studio 1
 - Installing Android Studio 2
 - Using Android Studio 4
 - Starting a New Project 6
- Standalone SDK Tools 9
- Android Device Emulation 10
 - Android Virtual Device 11
 - GenyMotion 12
 - Xamarin Android Player 13
- Version-Control Systems 14
 - Subversion 14
 - Git 14
 - Mercurial 15
- Summary 15

2 Testing and Debugging 17

- Unit Testing 17
- Integration Testing 20
- Debugging 25
 - Profiling 25
 - Tracing 27
 - Messaging 29
- Summary 32

3 Application Structure 33

- Manifests 34
- Java 36
- Res (Resources) 37
 - Drawable 37
 - Layout 39
 - Menu 39

Values	40
Other Resources	41
Gradle	41
Summary	42
4 Components	45
Intents	45
Intent Filters	46
Broadcast Receivers	47
Activities	48
Creating an Activity	48
Activity Lifecycle	49
Fragments	52
Creating a Fragment	52
Communicating with Fragments	55
Loaders	56
Summary	58
5 Views	59
The View Class	59
The AnalogClock Subclass	60
The ImageView Subclass	60
The KeyboardView Subclass	60
The MediaRouteButton Subclass	62
The ProgressBar Subclass	62
The Space Subclass	64
The SurfaceView Subclass	64
The TextView Subclass	65
The TextureView Subclass	65
The ViewGroup Subclass	66
The ViewStub Subclass	68
Creating a Custom View	68
Summary	70

6 Layout 71

- Layout Basics 71
 - Layout Measurements 72
 - Layout Coordinates 73
- Layout Containers 74
 - Linear Layout 74
 - Relative Layout 76
 - Table Layout 79
 - Frame Layout 80
 - WebView 82
- Summary 83

7 App Widgets 85

- App Widget Layouts 86
- The `AppWidgetProviderInfo` Object 88
 - App Widget Sizing 89
 - Update Frequency 90
 - Preview Image 90
 - Widget Category 92
 - Widget Category Layout 92
 - Resizable Mode 93
 - Sample `AppWidgetProviderInfo` Object 93
- The `AppWidgetProvider` Class 94
- Application Manifest Entries 96
- Summary 97

8 Application Design: Using MVC 99

- Model 100
- View 101
- Controller 102
- Working Asynchronously 104
 - `AsyncTask` 105
- Summary 106

9 Drawing and Animation 107

- Graphics 107
 - Bitmaps 107
 - NinePatch 109
 - Drawables 111
 - OpenGL ES 114
- Animation 117
 - View Animation 117
 - Property Animation 118
 - Drawable Animation 122
 - Transition Framework 123
- Summary 125

10 Networking 127

- Accessing the Internet 127
 - Network Detection 127
 - Using an HTTP Client 129
- Parsing XML 131
- Handling Network Operations Asynchronously 133
- Volley 135
- Summary 138

11 Working with Location Data 139

- Permissions 139
- Google Play Services Locations API 148
- Summary 153

12 Multimedia 155

- Working with Audio 155
 - Audio Playback 156
 - Audio Recording 159
- Working with Video 161
 - Video Playback 162
- Summary 165

13 Optional Hardware APIs 167

- Bluetooth 167
 - Enabling Bluetooth 168
 - Discovering Devices with Bluetooth 169
 - Connecting via Bluetooth Classic 171
 - Communicating with BLE 173
- Near Field Communication 176
 - ACTION_NDEF_DISCOVERED 177
 - ACTION_TECH_DISCOVERED 178
 - ACTION_TAG_DISCOVERED 179
- Device Sensors 181
 - Detecting the Available Sensors 182
 - Reading Sensor Data 183
- Summary 185

14 Managing Account Data 187

- Getting Accounts 187
- Android Backup Service 188
- Using Google Drive Android API 191
- Using Google Play Games Services 195
 - Working with Saved Games 196
- Summary 199

15 Google Play Services 201

- Adding Google Play Services 201
- Using Google API Client 203
- Google Fit 207
 - Enable API and Authentication 207
 - App Configuration and Connection 208
- Nearby Messages API 209
 - Enabling Nearby Messages 209
 - Sending and Receiving Messages 210
- Summary 214

16 Android Wear 217

- Android Wear Basics 217
- Screen Considerations 218

- Debugging 221
 - Connecting to an Emulator 221
 - Connecting to a Wear Device 222
- Communicating with Android Wear 224
 - Notifications 224
 - Sending Data 226
- Summary 228

17 Google Analytics 229

- Adding Google Analytics 229
- Google Analytics Basics 232
 - Events 233
 - Goals 234
 - Ecommerce 235
 - Custom Timings 235
 - Custom Dimensions 236
 - Custom Metrics 236
- Summary 237

18 Optimization 239

- Application Optimization 239
 - Application First 239
 - Application Logging 241
 - Application Configuration 242
- Memory Management 243
 - Garbage Collection Monitoring 245
 - Checking Memory Usage 245
- Performance 247
 - Working with Objects 247
 - Static Methods and Variables 248
 - Enhanced for Loops 248
 - float, double, and int 249
 - Optimized Data Containers 249
- Summary 249

19 Android TV 251

- The Big Picture 251
 - Ten-Foot View 252
 - TV Capabilities 254
 - Text, Color, and Bitmaps 255
- Building an App 258
- Emulation and Testing 261
- Summary 263

20 Application Deployment 265

- Preparing for Deployment 265
- Production Checklist 266
 - Certificate Keys 266
 - Contact Email 266
 - App Website 267
 - External Services or Servers 267
 - Application Icon 267
 - Licensing 268
 - Appropriate Package Name 268
 - Verifying Permissions and Requirements 269
 - Log and Debug Removal 270
 - Removal of Excess Unused Assets 270
- Preparing for Google Play 270
 - Application Screenshots 271
 - Promo Video 271
 - High-Res Icon 271
 - Feature Graphic 272
 - Promo Graphic 272
 - Banner for Android TV 272
 - Getting Paid 272
- APK Generation 273
- Summary 274

Index 275

Preface

The growth of Android since the launch of Cupcake has been astonishing. Today, Android powers more than just mobile phones; it has become the go-to solution for manufacturers of audio equipment, tablets, televisions, cars, and more.

As the use of Android becomes more prevalent, the demand for developers who are familiar with using it has also scaled. Developers who understand how the system can be built, leveraged, and used are necessary to provide the next wave of amazing and must-have applications.

Many people around the world are being introduced to Android for the first time, and we as developers need to make sure to provide them with a first-class experience that will put a smile on their face and help them understand how truly amazing the Android system is.

Why Development Patterns?

In the fast-paced world of development, patterns are the time-saving solutions that developers use and access to maximize their output and minimize time wasted creating a solution that will ultimately fail.

Android development is a special place that is both familiar and foreign to many Java and object-oriented programmers. The relationship it has with the Java language and structure helps to bring in developers who have experience and get them up to speed in an almost effortless manner. However, there are some optimizations and memory-handling techniques that are not optimal for the seasoned Java developer.

This particular book is the bridge that helps seasoned developers understand the Android way of building and thinking. It is written so that those new to Android development gain a foundation for the platform and how to work with the many facets and intricacies that Android brings to the table while giving some in-depth hints and strategies that advanced developers will need to make their app a success.

Who Should Read This Book?

Anyone interested in how Android development works should find this book enjoyable and helpful. Those just beginning their Android journey may not find this as complete of a volume, but some development experience will help; however, those who are tenacious and don't mind getting elbows-deep should find this to be an acceptable companion on their quest toward their perfect app.

Those who are interested in seeing only theoretical development patterns with large explanations about individual bit-shifting and hand-tuning memory management will be disappointed in that this book instead focuses on how Android works together piece-by-piece with example snippets that help solidify how things should be accomplished in a best-practices manner.

Getting Started

For those new to developing Android applications, the minimum requirement is a computer running either OS X, Windows, or Linux. On these systems, you should download Android Studio from <http://developer.android.com/sdk/>. Android Studio comes with the Android SDK.

Full use of the Android SDK requires downloads of the version and sample code for which you want to develop. Although you can certainly download only a specific version of Android, you should download all versions of Android on which you want your app to work.

You should also use the Android SDK to download system images of emulators or Android Virtual Device (AVD) files. These system images allow you to test your app without actually having an Android device.

It is highly recommended that you acquire at least one Android device for testing, with a preference of having multiple devices in many form-factors so that you can accurately test, monitor, and experience your app as your users will.

Visit the following websites to keep up on Android and see when new features are introduced and how to use them:

- **StackOverflow:** <http://www.stackoverflow.com/>
- **Official Android Developer Site:** <http://developer.android.com/>
- **Android Developers Blog:** <http://android-developers.blogspot.com/>
- **Google Developers on YouTube:** <https://www.youtube.com/user/androiddevelopers>
- **Android Source Code (AOSP):** <http://source.android.com/>

Book Structure

This book starts with the basics of Android development, including how to set up an environment. It takes you through the importance of creating a proper development flow and adding testing to your app to make sure your code performs and behaves the way you expect.

It continues step by step through the various pieces and parts that make up the Android framework. This includes how applications are structured, using widgets and components, and learning how to use and create views.

You are then introduced to application design paradigms and learn how to make sure you are creating an app that you can manage and update easily. This includes adding media and network connections that will not end up wasting precious battery power and giving users the most accurate and up-to-date information possible.

Optional hardware components, Android Wear, and Android TV are also covered later in this book to expose you to taking your app to the next level and exploring new opportunities. As Android finds itself being included in more devices, you'll understand how and why it is in your best interest to provide apps to users who invest in these platforms.

Finally, you learn about some key optimization strategies as well as how to package your app for distribution through enterprise systems, email, and the Google Play Store.

When you are finished with this book, you will have an understanding of how the Android system works and, more importantly, how to craft an app that is optimized, distributed, and enjoyed by what will hopefully be millions of users.

Register your copy of *Android Development Patterns* at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN **9780133923681** and click **Submit**. Once the process is complete, you will find any available bonus content under “Registered Products.”

Acknowledgments

Creating a book is a monumental effort that is never accomplished without the help, effort, guidance, and diligence of a small band of heroes. I could never have completed this work without the correction of three of the greatest technical editors in the field today. Massive thanks, a hat-tip, and cheers go to Romin Irani, Douglas Jones, and Ray Rischpater for each bringing a personal penchant of perfection to the book and making sure I didn't stray too far off the established path.

I also give an enthusiastic thanks to my development editor, Sheri Replin. Sheri has been great to work with, and she tolerates the brief moments of madness I have where I am certain that the words I have chosen make complete sentences when they are actually the inane babble of a caffeine-deprived developer. Also, credit is due to my amazing copy editor, Bart Reed. He miraculously managed to properly apply a clever and intelligent sheen to my stark ravings, making the book read as it originally sounded in my head, as well as making it clear to the reader.

As always, the world-class team at Pearson deserves more thanks than I believe they get. Specifically, I would like to call out Laura Lewin, Olivia Basegio, Elaine Wiley, Kristy Hart, Mark Taub, and the entire production staff. The steps that are taken to create these volumes of technical instruction do not happen overnight, and these fine folks have undergone hours of meetings, emails, phone calls, and more to make sure that you get the greatest-and-latest book possible.

I want to thank my family for letting me disappear almost every night and every weekend for the past year. It has been an epic struggle keeping the book on schedule, working a sometimes more-than-full-time job, and also making sure that I attend the activities that matter most with them. I believe that it is all of you who have let me keep a pretty good work-life-book balance.

Finally, I thank you! Thank you for picking up this book and giving it a place on your shelf (digital or otherwise). With all the amazing people I have had the opportunity to work with, I believe we have crafted a book that will get you on the best path to creating Android applications that will be used for years to come.

About the Author

Phil Dutson is a Solution Architect over client-side and mobile implementation for one of the world's largest e-commerce retailers in fitness equipment. He has been collecting and developing for mobile devices since he got his hands on a US Robotics Pilot 5000. He is the author of *Sams Teach Yourself jQuery Mobile in 24 Hours* (Sams, July 2012), *jQuery*, *jQuery UI*, and *jQuery Mobile: Recipes and Examples* (Pearson, November 2012), *Android Developer's Cookbook, Second Edition* (Pearson, July 2013), and *Responsive Mobile Design* (Addison-Wesley Professional, September 2014).

Views

Of all the pieces of the Android system, views are probably the most used. Views are the core building block on which almost every piece of the UI is built. They are versatile and, as such, are used as the foundation for widgets. In this chapter, you learn how to use and how to create your own view.

The view Class

A view is a rather generic term for just about anything that is used in the UI and that has a specific task. Adding something as simple as a button is adding a view. Some widgets, including `Button`, `TextView`, and `EditText` widgets, are all different views.

Looking at the following line of code, it should stand out that a button is a view:

```
Button btnSend = (Button) findViewById(R.id.button);
```

You can see that the `Button` object is defined and then set to a view defined in the application layout XML file. The `findViewById()` method is used to locate the exact view that is being used as a view. This snippet is looking for a view that has been given an `id` of `button`. The following shows the element from the layout XML where the button was created:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true" />
```

Even though the element in the XML is `<Button>`, it is still considered a view. This is because `Button` is what is called an indirect subclass of `View`. In total, there are more than 80 indirect subclasses of `View` as of API level 21. There are 11 direct subclasses of `View`: `AnalogClock`, `ImageView`, `KeyboardView`, `MediaRouteButton`, `ProgressBar`, `Space`, `SurfaceView`, `TextView`, `TextureView`, `ViewGroup`, and `ViewStub`.

The AnalogClock Subclass

The `AnalogClock` is a complex view that shows an analog clock with a minute-hand and an hour-hand to display the current time.

Adding this view to your layout XML is done with the following element:

```
<AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/analogClock"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
```

This view can be attached to a surface by using the `onDraw(Canvas canvas)` method, and it can be sized to scale to the screen it is being displayed on via the following method:

```
onMeasure(int widthMeasureSpec, int heightMeasureSpec)
```

It should be noted that if you decide to override the `onMeasure()` method, you must call `setMeasuredDimension(int, int)`. Otherwise, an `IllegalStateException` error will be thrown.

The ImageView Subclass

The `ImageView` is a handy view that can be used to display images. It is smart enough to do some simple math to figure out dimensions of the image it is displaying, which in turn allows it to be used with any layout manager. It also allows for color adjustments and scaling the image.

Adding an `ImageView` to your layout XML requires the following:

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/car"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
```

To show multiple figures, you can use multiple `ImageViews` within a layout. Similar to other views, you can attach events such as a click event to trigger other behavior. Depending on the application you are building, this may be advantageous versus requiring the user to click a button or use another widget to complete an action.

The KeyboardView Subclass

The `KeyboardView` is one of the most interesting views that exist. This is one of the true double-edged components of the Android system. Using the `KeyboardView` allows you to

create your own keyboard. Several keyboards exist in the Play store that you can download right now and use on your Android device that are based on using the `KeyboardView`.

The problem is that using an application with a custom keyboard means that all data entry must pass through it. Every “keystroke” is passed through the application, and that alone tends to send shivers down the spine of those who are security conscious. However, if you are an enterprise developer and need a custom keyboard to help with data entry, then this view may be exactly what you are looking for.

Note

The `KeyboardView` requires creating a new input type for your device, and the keyboard you create will be accessible in all programs. This also means that users may opt to not use your keyboard, and may even disable it as an option.

Creating your own keyboard is an involved process. You need to do the following:

- Create a service in your application manifest.
- Create a class for the keyboard service.
- Add an XML file for the keyboard.
- Edit your `strings.xml` file.
- Create the keyboard layout XML file.
- Create a preview `TextView`.
- Create your keyboard layout and assign values.

The `KeyboardView` has several methods you can override to add functionality to your keyboard:

- `onKey()`
- `onPress()`
- `onRelease()`
- `onText()`
- `swipeDown()`
- `swipeUp()`
- `swipeLeft()`
- `swipeRight()`

You do not need to override all of these methods; you may find that you only need to use the `onKey()` method.

The `MediaRouteButton` Subclass

The `MediaRouteButton` that is part of the compatibility library is generally used when working with the Cast API. This is where you need to redirect media to a wireless display or ChromeCast device. This view is the button that is used to allow the user to select where to send the media.

Note that per Cast design guidelines, the button must be considered “top level.” This means that you can create the button as part of the menu or as part of the `ActionBar`. After you create the button, you must also use the `.setRouteSelector()` method; otherwise, an exception will be thrown.

First, you need to add an `<item>` to your menu XML file. The following is a sample `<item>` inside of the `<menu>` element:

```
<item
  android:id="@+id/mediaroutebutton_cast"
  android:actionProviderClass="android.support.v7.app.MediaRouteActionProvider"
  android:actionViewClass="android.support.v7.app.MediaRouteButton"
  android:showAsAction="always"
  android:visible="false"
  android:title="@string/mediaroutebutton"/>
```

Now that you have a menu item created, you need to open your `MainActivity` class and use the following import:

```
import android.support.v7.app.MediaRouteButton;
```

Next, you need to declare it in your `MainActivity` class:

```
private MediaRouteButton myMediaRouteButton;
```

Finally, add the code for the `MediaRouteButton` to the menu of the `onCreateOptionsMenu()` method. Remember that you must also use `setRouteSelector()` on the `MediaRouteButton`. The following demonstrates how this is accomplished:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    getMenuInflater().inflate(R.menu.main, menu);

    myMediaRouteItem = menu.findItem(R.id.mediaroutebutton_cast);
    myMediaRouteButton = (MediaRouteButton) myMediaRouteItem.getActionView();
    myMediaRouteButton.setRouteSelector(myMediaRouteSelector);
    return true;
}
```

The `ProgressBar` Subclass

The progress bar is a familiar UI element. It is used to indicate that something is happening and how far along this process is. It is not always possible to determine how long an action will

take; luckily, the `ProgressBar` can be used in indeterminate mode. This allows an animated circle to appear that shows movement without giving a precise measurement of the status of the load.

To add a `ProgressBar`, you need to add the view to your layout XML. The following shows adding a “normal” `ProgressBar`:

```
<ProgressBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/progressBar"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
```

Other styles of `ProgressBar` may also be used. To change the style, you need to add a property to the `<ProgressBar>` element. The following styles may be used:

```
Widget.ProgressBar.Horizontal
Widget.ProgressBar.Small
Widget.ProgressBar.Large
Widget.ProgressBar.Inverse
Widget.ProgressBar.Small.Inverse
Widget.ProgressBar.Large.Inverse
```

Depending on your implementation, you may apply the style either with your `styles.xml` or from your `attrs.xml`. For the styles from `styles.xml`, you would use the following:

```
style="@android:style/Widget.ProgressBar.Small"
```

If you have styles inside your `attrs.xml` file that you want applied to the progress bar, use the following property in the `<ProgressBar>` element:

```
style="?android:attr/progressBarStyleSmall"
```

If you are planning on using the indeterminate mode, you need to pass a property of `android:indeterminate` into the `<ProgressBar>` element. You may also specify the loading animation by setting the `android:indeterminateDrawable` to a resource of your choosing.

A `ProgressBar` that is determinate requires updates to be passed to it via the `setProgress()` or `incrementProgressBy()` method. These methods should be called from a worker thread. The following shows an example of a thread that uses a `Handler` and an `int` for keeping the progress value, and a `ProgressBar` has been initialized:

```
new Thread(new Runnable() {
    public void run() {
        while (myProgress < 100) {
            myProgress = doWork();
            myHandler.post(new Runnable() {
                public void run() {
                    myProgressBar.setProgress(myProgress);
                }
            });
        }
    }
});
```



```

        });
    }
}
}).start();

```

The Space Subclass

For those who have worked on layouts and visual interfaces, the `Space` view is one that is both helpful and brings on somewhat lucid nightmares. This view is reserved to add “space” between other views and layout objects.

The benefit to using a `Space` is that it is a lightweight view that can be easily inserted and modified to fit your needs without you having to do an absolute layout or extra work trying to figure out how relative spacing would work on complex layouts.

Adding a `Space` is done by adding the following to your layout XML:

```

<Space
    android:layout_width="1dp"
    android:layout_height="40dp" />

```

The SurfaceView Subclass

The `SurfaceView` is used when rendering visuals to the screen. This may be as complex as providing a playback surface for a live camera feed, or it can be used for rendering images on a transparent surface.

The `SurfaceView` has two major callbacks that act as lifecycle mechanisms that you can use to your advantage: `SurfaceHolder.Callback.surfaceCreated()` and `SurfaceHolder.Callback.surfaceDestroyed()`. The time in between these methods is where any work with drawing on the surface should take place. Failing to do so may cause your application to crash and will get your animation threads out of sync.

Adding a `SurfaceView` requires adding the following to your layout XML:

```

<SurfaceView
    android:id="@+id/surfaceView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1" />

```

Depending on how you are going to use your `SurfaceView`, you may want to use the following callback methods:

- `surfaceChanged()`
- `surfaceCreated()`
- `surfaceDestroyed()`

Each of these callback methods gives you an opportunity to initialize values, change them, and more importantly free some system resources up when it is released. If you are using a `SurfaceView` for rendering video from the device camera, it is essential that you release control of the camera during the `surfaceDestroyed()` method. Failing to release the camera will throw errors when you attempt to resume usage of the camera in either another application or when your application is resumed. This is due to a new instance attempting to open on a resource that is finite and currently marked as in use.

The `TextView` Subclass

The `TextView` is likely the first view added to your project. If you create a new project in Android Studio that follows the default options, you will be given a project that contains a `TextView` with a string value of “Hello World” in it.

To add a `TextView`, you need to add the following code to your layout XML file:

```
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Note that in the previous example, the value for the `TextView` is taken from `@string/hello_world`. This value is inside of the `strings.xml` file that is in your `res/values` folder for your project. The value is defined in `strings.xml` as follows:

```
<string name="hello_world">Hello world!</string>
```

The `TextView` also contains a large number of options that can be used to help format, adjust, and display text in your application. For a full list of properties, visit <http://developer.android.com/reference/android/widget/TextView.html>.

The `TextureView` Subclass

The `TextureView` is similar to the `SurfaceView` but carries the distinction of being tied directly to hardware acceleration. OpenGL and video can be rendered to the `TextureView`, but if hardware acceleration is not used for the rendering, nothing will be displayed. Another difference when compared to `SurfaceView` is that `TextureView` can be treated like a `View`. This allows you to set various properties including setting transparency.

In similarity to `SurfaceView`, some methods need to be used with `TextureView` in order for proper functionality. You should first create your `TextureView` and then use either `getSurfaceTexture()` or `TextureView.SurfaceTextureListener` before using `setContentProvider()`.

Callback methods should also be used for logic handling while working with the `TextureView`. Paramount among these callback methods is the `onSurfaceTextureAvailable()` method. Due to `TextureView` only allowing one content provider to manipulate it at a time, the

`onSurfaceTextureAvailable()` method can allow you to handle IO exceptions and to make sure you actually have access to write to it.

The `onSurfaceTextureDestroyed()` method should also be used to release the content provider to prevent application and resource crashing.

The ViewGroup Subclass

The `ViewGroup` is a special view that is used for combining multiple views into a layout. This is useful for creating unique and custom layouts. These views are also called “compound views” and, although they are flexible, they may degrade performance and render poorly based on the number of children included, as well as the amount of processing that needs to be done for layout parameters.

CardView

The `CardView` is part of the `ViewGroup` that was introduced in Lollipop as part of the v7 support library. This view uses the Material design interface to display views on “cards.” This is a nice view for displaying compact information in a native Material style. To use the `CardView`, you can load the support library and wrap your view elements in it. The following demonstrates an example:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <android.support.v7.widget.CardView
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:id="@+id/card_view"
        android:layout_gravity="center"
        android:layout_width="200dp"
        android:layout_height="200dp"
        card_view:cardCornerRadius="4dp"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true">

        <TextView android:text="@string/hello_world"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </android.support.v7.widget.CardView>
</RelativeLayout>
```

This example shows a card in the center of the screen. The color and corner radius can be changed via attributes in the `<android.support.v7.widget.CardView>` element. Using `card_view:cardBackgroundColor` will allow you to change the background color, and using `card_view:cardCornerRadius` will allow you to change the corner radius value.

Note

Using the `CardView` support library requires you to edit your Gradle build files. You need to add the following line to the dependencies section in your `build.gradle` file:

```
dependencies {
    compile 'com.android.support:cardview-v7:21.+'
```

You should change the version number targeted on the end to match your project target.

RecyclerView

The `RecyclerView` was also added in Lollipop as part of the v7 support library. This view is a replacement for the aging `ListView`. It brings with it the ability to use a `LinearLayoutManager`, `StaggeredLayoutManager`, and `GridLayoutManager` as well as animation and decoration support. The following shows how you can add this view to your layout XML:

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/my_recycler_view"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Similar to with a `ListView`, after you have added the `RecyclerView` to your layout, you then need to instantiate it, connect it to a layout manager, and then set up an adapter to display data.

You instantiate the `RecyclerView` by setting it up as follows:

```
myRecyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);
```

The following shows connecting to a layout manager using the `LinearLayoutManager` that is part of the v7 support library:

```
myLayoutManager = new LinearLayoutManager(this);
myRecyclerView.setLayoutManager(myLayoutManager);
```

All that is left is to attach the data from an adapter to the `RecyclerView`. The following demonstrates how this is accomplished:

```
myAdapter = new MyAdapter(myDataset);
myRecyclerView.setAdapter(myAdapter);
```

The ViewStub Subclass

The `ViewStub` is a special view that is used to create views on demand in a reserved space. The `ViewStub` is placed in a layout where you want to place a view or other layout elements at a later time. When the `ViewStub` is displayed—either by setting its visibility with `setVisibility(View.VISIBLE)` or by using the `inflate()` method—it is removed and the layout it specifies is then injected into the page.

The following shows the XML needed to include a `ViewStub` in your layout XML file:

```
<ViewStub
    android:id="@+id/stub"
    android:inflatedId="@+id/panel_import"
    android:layout="@layout/progress_overlay"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom" />
```

When the `ViewStub` is inflated, it will use the layout specified by the `android:layout` property. The newly inflated view will then be accessible via code by the ID specified by the `android:inflatedId` property.

Creating a Custom View

When developing your own application, you may need a view that doesn't come "out of the box." When this occurs you have two options: You can create a class for your own custom view or you may extend one of the existing views.

To create your own, you need to create a new class, have it extend `View`, and have it override at least one method. You will also be adding the variables and logic needed to handle the custom properties you will be adding to your view. The following shows a custom view along with the values used as custom properties:

```
public class MyView extends View {
    private int viewColor, viewBgColor;

    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);

        TypedArray a = context.getTheme().obtainStyledAttributes(attrs,
            R.styleable.MyView, 0, 0);

        try {
            viewColor = a.getInteger(R.styleable.MyView_viewColor);
            viewBgColor = a.getInteger(R.styleable.MyView_viewBgColor)
        } finally {
            a.recycle();
        }
    }
}
```

```

    @Override
    protected void onDraw(Canvas canvas) {
        // draw your view
    }
}
}
}

```

You want to be able to pass values through the XML when used with your application layout XML. To do this you can add an XML file to the `res/values` folder. This folder houses `<resources>` with child `<declare-styleable>` elements. The following shows an example of a custom view XML file:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="MyView">
        <attr name="viewColor" />
        <attr name="viewBgColor" />
    </declare-styleable>
</resources>

```

Now you can add your custom view to your application layout, but you need to add a property so that your custom view can be found. This is done by adding the following line to your layout element:

```
xmlns:custom="http://schemas.android.com/apk/res/com.dutsonpa.mycustomview"
```

Notice that you need to change the value to match your namespace by replacing `com.dutsonpa.myview` with your own package name. Once you add that to your layout element, you can add your custom view. This is done by referencing the package and then adjusting or setting the values you want to use. The following shows an example of a custom view being added with values being set:

```

<com.dutsonpa.mycustomview.myview
    android:id="@+id/"
    custom:viewColor="#33FF33"
    custom:viewBgColor="#333333" />

```

Notice that Android properties may be used and that your custom properties are used by employing `custom:valueName`. This provides some flexibility by allowing some built-in features to be mixed with your custom attributes.

The last thing you should do is add getter and setter methods for your attributes. These can be added to your class as follows:

```

public void getViewColor() {
    return viewColor;
}

```

```
public void getViewBgColor() {
    return viewBgColor;
}

public void setViewColor(int newViewColor) {
    viewColor=newViewColor;
    invalidate();
    requestLayout();
}

public void setViewBgColor(int newViewBgColor) {
    viewBgColor=newViewBgColor;
    invalidate();
    requestLayout();
}
```

By using `invalidate()` and `requestLayout()`, the layout is forced to redraw using the `onDraw()` method that is being employed by the custom view.

Summary

In this chapter, you learned what views are and how they are used in applications. You learned that views have multiple subclasses that can be used as is or extended by making a custom `View`.

You learned about the main subclasses and how to implement them into your application layout XML file, as well as some code that may be used to accompany them.

You also learned about two views that were introduced with Android Lollipop: `CardView` and `RecyclerView`. These views are complex `ViewGroups` that can help display data in the Material design style and update the aging `ListView`.

Index

A

ACCESS_COARSE_LOCATION permission, 139-140

ACCESS_FINE_LOCATION permission, 140

ACCESS_NETWORK_STATE permission, 127

accessing

content providers, 100

Internet, 127-130

HTTP clients, 129-130

mobile data connectivity,
detecting, 128

network detection, 127-128

Wi-Fi connectivity, detecting, 128

AccountManager class, 187

**ACTION_NDEF_DISCOVERED intent,
filtering, 177-178**

**ACTION_TAG_DISCOVERED intent, filtering,
179-181**

**ACTION_TECH_DISCOVERED intent, filtering,
178-179**

Activities, 48-51

adding to manifests folder, 49

callback methods, 50-51

creating, 48-49

Fragments, adding, 55-56

lifecycle, 49-51, 101-102

Loaders, 56-57

Location API, 148-153

location reporting application, 140-144

Activity-selection screen (Android Studio), 6

ADT (Android Development Tools), 1

migrating to Android Studio, 3-4

advertisement services for Android TV, 253

alpha, 118

AnalogClock view, 60

android avd command, 11

Android Backup Services, 187-191

Backup Service Key, 189-190

preferences, backing up, 190-191

Android Developers Blog, xv

Android developers page (YouTube), xv

**Android Device Monitor, application
profiling, 26-27**

Android Player, website, 13

Android SDK, xv

downloading, 9

Android SDK Manager, 10

Android Source Code website, xv

Android Studio, 1-2

Activity-selection screen, 6

Android TV apps, building, 258-261

AVD Manager, launching, 11

build problems, troubleshooting, 6-7

Build Variants window, 19

Design view, 7

features, 8-9

Gradle build system, 2

installing, 2-4

migration from ADT, 3-4

Preview pane, 8

projects

closing, 9

Gradle build system, 41-42

java folder, 36-37

manifests folder, 34-36

starting, 4-9

testing folders, creating, 18

Recent Projects list, 4

website, xv, 2

Welcome Screen, 4

Android TV, 251-252. See also TVs

advertisement services, 253

app banners, 256-257, 272

apps, building, 258-261

bitmaps, 258

controls, 253

debugging, 261-262

device emulators, 262

Focused state, 253

guidelines

color, 256

text, 255-256

Leanback, 252

LinearLayout, 254

recommendations, 257-258

ten-foot view, 252-254

web resources, 253

widgets, 258

Android Wear, 217-218

BoxInsetLayout class, 219-220

communicating with, 224-228

notifications, 224-226

sending data, 226-228

comparing Wear and Android
devices, 218

devices, connecting to, 222-224

emulators, 221-222

WatchViewStub class, 218-219

Wearable UI Library, 218

animation, 117-125

- drawable animation, 122
 - defining in XML, 122
- property animation, 118-121
 - AnimatorSet subclass, 121
 - defining in XML, 119
 - ObjectAnimator subclass, 120-121
 - ValueAnimator subclass, 120
- scale effect, 118
- transition framework, 123-125
 - defining in XML, 123-124
 - fade out/in transitions, 125
- view animation, 117-118
 - alpha, 118
 - creating, 118
 - interpolators, 118
 - rotate effect, 118
 - translate effect, 118

AnimatorSet subclass, 121**annotations, 19****ANR (Application Not Responding) notices, 21****APIs**

- Bluetooth
 - enabling, 168-169
 - scanning for devices, 169-170
 - stages of communication, 167-168
- device sensors, 181-185
 - detecting, 182-183
 - reading data, 183-185
- Google Drive Android API, 191-195
 - client, creating, 191-192
 - reading files, 193-194
 - registering applications with Developers Console, 191
 - retrieving files, 193
 - writing to files, 194-195

Google Fit, 207

Location API, 148-153

Nearby API, 209-214

Nearby Messages API

- enabling, 209-210

- sending and receiving messages, 210-214

NFC, 176-181

- ACTION_NDEF_DISCOVERED intent, filtering, 177-178

- ACTION_TAG_DISCOVERED intent, filtering, 179-181

- ACTION_TECH_DISCOVERED intent, filtering, 178-179

NDEF messages, 176

- permissions, 177

- tags, 176-177

app widgets, 85-86

- adding to lock screen, 92

Android TV, 258

AppWidgetProvider class, 94-96

AppWidgetProviderInfo object, 89-93

- creating, 85-86

- layouts, 86-89

- home screen, 92-93

- sizing, 89-90

- XML layout file, 86-89

- manifest file entries, 96

- preview image, 90-92

- resizing, 93

- update frequency, 90

Appium, 20**Application object, extending, 239-241****applications**

- Android TV apps, building, 258-261

- asynchronous processing, 104-106

- components
 - Activities, 48-51
 - Fragments, 52-57
 - Intents, 45-48
- deployment
 - preparing for, 265
 - production checklist, 266-270
- Google Services, enabling, 229-230
- location reporting
 - Activity, 140-148
 - layouts, 147-148
 - services, 144-146
- optimizing
 - Application object, extending, 239-241
 - logging, 241-242
 - versioning your configuration, 242-243
- profiling, 25-27
- registering with Android Backup Service, 189
- registering with Developers Console, 191
- AppWidgetProvider class, 94-96**
 - callback methods, 94
 - methods, 95-96
- AppWidgetProviderInfo object, 89-93**
- aspect ratio, 254**
- Assert, What a Terrible Failure log level, 30**
- asynchronous processing, 104-106**
 - AsyncTask, 105-106
 - worker threads, 104-105
- AsyncTask, 105-106, 133-135**
 - network handling, 133-135
- attributes, manifest file, 35**

- audio**
 - playback, 156-159
 - recording, 159-161
 - supported codecs, 155
- authentication, Google Fit, 207-208**
- available memory, displaying, 27**
- AVD (Android Virtual Device), 3, 11-12**
 - scaling, 262
 - virtual devices
 - cloning, 11
 - scaling, 11
- AVD Manager**
 - launching, 11
 - Wear emulator, creating, 221-222

B

- backing up**
 - preferences, 190-191
 - user data, 188-191
- Backup Service Key, 189-190**
- bitmaps, 107-111**
 - memory usage, 107-108
 - NinePatch, 109-111
 - Draw 9-patch utility, 110-111
 - scaling, 108-109
- BLE (Bluetooth low energy), 167**
 - communicating with, 173-176
 - GATT, 173
 - scanning for devices, 170
- blogs, Android Developers Blog, xv**
- Bluetooth**
 - BLE
 - communicating with, 173-176
 - GATT, 173
 - discovery stage, 167
 - enabling, 168-169

- exploration stage, 168
- Generic Access Profile, 167
- interaction stage, 168
- scanning for devices, 169-170
- Bluetooth Classic, 167, 171-173**
- bound state (services), 103**
- BoxInsetLayout class, 219-220**
- broadcast receivers, 47-48**
- build problems, troubleshooting in Android Studio, 6-7**
- Build Variants window (Android Studio), 19**
- build.gradle files, 42**
- Business license (GenyMotion), 13**
- buttons, 59**

C

callback methods

- Activities, 50-51
- AppWidgetProvider class, 94
- cancelDiscovery() method, 169**
- capturing heap dumps, 26-27**
- CardView view, 66-67**
- Cast API, MediaRouteButton, 62**
- cells, sizing widgets, 89-90**
- Certificate Authorities, 130**
- CheckUrlTask class, 134**
- child elements, aligning, 74-76**

classes

- AccountManager, 187
- AppWidgetProvider, 94-96
- AppWidgetProvider class
 - callback methods, 94
 - methods, 95-96
- BoxInsetLayout class, 219-220
- CheckUrlTask, 134
- singletons, 137

- SoundPool, 156-159
- WatchViewStub class, 218-219

cloning virtual devices, 11

closing

- Android Studio projects, 9
- HTTP connections, 129

coarse location data, permissions, 139-140

code repositories

- Git, 14-15
- Mercurial, 15
- Subversion, 14

codecs

- supported audio codecs, 155
- supported video codecs, 161

CollabNet, 14

color, guidelines for Android TV, 256

columns, table layout, 79-81

commands, android avd, 11

communicating

- with Android Wear, 224-228
 - notifications, 224-226
 - sending data, 226-228
- with BLE, 173-176
- with Fragments, 55-56

comparing Wear and Android devices, 218

components

- Activities, 48-51
 - callback methods, 50-51
 - creating, 48-49
 - lifecycle, 49-51
- Fragments, 52-57
 - adding to Activities, 55-56
 - communicating with, 55-56
 - creating, 52-55
 - lifecycle, 53-55
 - methods, 52-53

Intent filters, 46-47

Intents, 45-48

 broadcast receivers, 47-48

 explicit Intents, 45

 implicit Intents, 46

Loaders, 56-57

connecting to the Internet, 127-130

 HTTP clients, 129-130

 network detection, 127-128

 Wi-Fi connectivity, detecting, 128

ConnectivityManager, 128

containers, 249

 layouts

 frame layout, 81-83

 linear layout, 74-77

 relative layout, 77-79

 table layout, 79-81

content providers, 100-101

 accessing, 100

 deleting data from, 101

 inserting data, 100

 updating, 100

Controller (MVC architecture), 102-104

controlling layouts, 71

CPU, profiling, 25-27

creating

 Activities, 48-49

 broadcast receivers, 47-48

 custom views, 68-70

 Fragments, 52-55

 Intents

 explicit Intents, 46

 implicit Intents, 46

 location reporting application, 140-148

 Activity, 140-144

 layouts, 147-148

 services, 144-146

 queues with Volley, 135-136

 view animation, 118

 widgets, 85-86

Cupcake, xiv

custom dimensions, 236

custom timings, 235-236

custom views, creating, 68-70

D

DDMS (Dalvik Debug Monitor Server), 29

Debug log level, 30

debugging, 25-32. See also testing

 Android TV, 261-262

 Android Wear, 222-224

 messaging, 29-32

 profiling, 25-27

 available memory, displaying, 27

 heap dumps, capturing, 26-27

 tracing, 27-28

declaration, manifest file, 34

deploying applications

 APK generation, 273-274

 Google Play Store

 Android TV banners, 272

 charging for your app, 272-273

 feature graphic, 272

 high-res icon, 271-272

 promo graphic, 272

 promo video, 271

 screenshots, 271

- preparing for, 265
- production checklist, 266-270
 - certificate keys, 266
 - contact email, 266
 - external services, 267
 - launcher icon, 267-268
 - licensing, 268
 - package name, 268-269
 - removal of excess unused assets, 270
 - verifying permissions, 269
- Design view (Android Studio), 7**
- detecting**
 - network connectivity, 127-128
 - mobile data connectivity, 128
 - Wi-Fi, 128
 - sensors, 182-183
- “Developer mode”, enabling on Android device, 25**
- development. See also debugging; emulators; testing**
 - ADT, 1
 - migrating to Android Studio, 3-4
 - Android
 - minimum requirements, xv
 - websites
 - Android Studio, 1-2
 - Activity-selection screen, 6
 - build problems, troubleshooting, 6-7
 - Build Variants window, 19
 - Design view, 7
 - features, 8-9
 - Gradle build system, 2
 - installing, 2-4
 - Preview pane, 8
 - projects, starting, 6-9
 - Recent Projects list, 4
 - website, 2
 - Welcome Screen, 4
 - Android TV apps, building, 258-261
 - Android Wear, 217-218
 - asynchronous processing, 104-106
 - Google Services, enabling, 229-230
 - JDKs, 3
 - MVC architecture, 99
 - content providers, 100-101
 - Controller, 102-104
 - services, 102-104
 - views, 101-102
 - patterns, xiv
- device emulators, 10-13**
 - Android TV, 262
 - Android Wear, 221-222
 - AVD, 11-12
 - GenyMotion, 12-13
 - licensing, 13
 - virtual devices
 - cloning, 11
 - profiling, 25-27
 - Xamarin Android Player, 13
- dimensions, sizing widgets, 89-90**
- discovery stage (Bluetooth), 167**
- documentation for OpenGL ES, 117**
- downloading**
 - Android SDK, 9
 - Oracle VM Virtual Box, 12
- dp (density-independent) pixels, 72-73**
- Draw 9-patch utility, 110-111**
- drawable animation, 122**
 - defining in XML, 122
- drawable folder, 37-38**

- drawables, 111-114**
 - primitive shapes, 111-113
 - system drawables, 113-114

E

- Eclipse IDE, 1**
- ecocommerce, 235**
- elements in manifest file, 35-36**
- emulators, 10-13**
 - Android TV, 262
 - Android Wear, 221-222
 - AVD, 11-12
 - GenyMotion, 12-13
 - licensing, 13
 - virtual devices
 - cloning, 11
 - heap dumps, capturing, 26-27
 - profiling, 25-27
 - scaling, 11
 - Xamarin Android Player, 13
- Error log level, 30**
- EULA (End User License Agreement), 268**
- events, 233-234**
- explicit Intents, 45**
 - creating, 46
- exploration stage (Bluetooth), 168**
- extending the Application object, 239-241**

F

- fade out/in transitions, 125**
- features**
 - of Android Studio, 8-9
 - of Git, 15
 - of Google Analytics, 232-233
 - of Mercurial, 15
 - of Subversion, 14

- file structure of projects**
 - java folder, 36-37
 - manifests folder, 34-36
 - Activities, adding, 49
 - res folder, 37-41
 - drawable subfolder, 37-38
 - layout subfolder, 39
 - menu subfolder, 39
 - values subfolder, 40-41
- fine location data, permissions, 140**
- Fitness API, enabling, 207-208**
- Focused state, 253**
- folders**
 - java folder, 36-37
 - manifests folder, 34-36
 - Activities, adding, 49
 - res folder, 37-41
 - drawable subfolder, 37-38
 - layout subfolder, 39
 - menu subfolder, 39
 - values subfolder, 40-41
 - testing folders, creating, 18
- for loops, 248-249**
- Fragments, 52-57**
 - adding to Activities, 55-56
 - communicating with, 55-56
 - creating, 52-55
 - lifecycle, 53-55
 - Loaders, 56-57
 - methods, 52-53
 - views, 55
- frame layout, 81-83**
 - overlays, 81-82
 - WebView view, 82-83

frameworks

- Robotium automation framework, 20
- transition framework, 123-125

Free license (GenyMotion), 13**functions**

- getMemoryClass(), 244
- helper functions, 74

funnels, 234**Fused Location Provider, 148**

G

game saving, adding to Google Play Games Services, 196-197**garbage collection**

- minimizing, 244
- monitoring, 245

GATT (Generic Attribute Profile), 173**Generic Access Profile, 167****GenyMotion, 12-13**

- licensing, 13

getActivity() method, 56**getLoaderManager() method, 56****getMemoryClass() function, 244****getResponseCode() method, 133****GLSurfaceView, setting up, 115-117****goals (Google Analytics), 234****Google accounts, retrieving from devices, 187-188****Google Analytics**

- available statistics, 232
- custom dimensions, 236
- custom metrics, 236-237
- custom timings, 235-236
- ecommerce, 235
- enabling, 230-232
- events, 233-234

features, 232-233

funnels, 234

goals, 234

Google API Client, connecting to Google Play Services, 203-206**Google Drive Android API, 191-195**

- applications, registering with Developers Console, 191
- client, creating, 191-192
- reading files, 193-194
- retrieving files, 193
- writing to files, 194-195

Google Fit, 207-209

- APIs, 207
- authentication, 207-208
- configuring, 208-209

Google Payments Merchant Center, 272-273**Google Play Games Services, 195-199**

- game saving, adding, 196-197
- loading saved games, 197-199
- Snapshots, 197

Google Play Services, 201

- adding to Gradle file, 202
- available services, 202
- compiling, 202
- connecting to, 203-206
- Google Fit, 207-209
 - APIs, 207
 - authentication, 207-208
 - configuring, 208-209
 - Fitness API, enabling, 207-208
- initial setup, 201
- Location API, 148-153
- Nearby API, 209-214

Nearby Messages API
 enabling, 209-210
 sending and receiving messages,
 210-214
 override methods, 204-206

Google Play Store, listing your app with

Android TV banners, 272
 charging for your app, 272-273
 feature graphic, 272
 high-res icon, 271-272
 promo graphic, 272
 promo video, 271
 screenshots, 271

Google Services, enabling, 229-230

Google (Android) TV, 251-252. See also TVs

advertisement services, 253
 app banners, 256-257, 272
 apps, building, 258-261
 bitmaps, 258
 controls, 253
 debugging, 261-262
 device emulators, 262
 Focused state, 253
 guidelines
 color, 256
 text, 255-256
 Leanback, 252
 LinearLayout, 254
 recommendations, 257-258
 ten-foot view, 252-254
 web resources, 253
 widgets, 258

GPS

coarse location data, permissions,
 139-140
 fine location data, permissions, 140

Gradle build system, 2, 41-42

build.gradle files, 42
 Google Play Services, adding, 202
 gradle.build file, adding support for
 JUnit, 18

graphics. See also images

bitmaps, 107-111
 memory usage, 107-108
 scaling, 108-109
 drawables, 111-114
 shapes, 111-113
 system drawables, 113-114
 NinePatch, 109-111
 OpenGL ES, 114-117
 adding to manifest file, 114-115
 documentation, 117
 GLSurfaceView, setting up, 115-117
 NDK, 117
 texture compression, 115

guidelines for Android TV

color, 256
 text, 255-256

H

hasResolution() method, 204

heap dumps, capturing, 26-27

helper functions, 74

home screen, widgets

adding, 92
 layouts, 92-93

Hovered state, 253

HTTP clients, 129-130

|

IDEs (integrated development environments). *See also* development

- Eclipse, 1
- emulators, 10-13
 - AVD, 11-12
 - GenyMotion, 12-13
 - Xamarin Android Player, 13
- JDKs, 3
- JetBrains IntelliJ IDEA, 1
 - website, 9

imageRequests, 137**images**

- bitmaps, 107-111
 - memory usage, 107-108
 - NinePatch, 109-111
 - scaling, 108-109
- drawables, 111-114
 - primitive shapes, 111-113
 - system drawables, 113-114

ImageView view, 60**implicit Intents, 46**

- creating, 46

Indie license (GenyMotion), 13**Info log level, 30****InformIT, book registration, xvi****installing Android Studio, 2-4**

- OS X, 2-3

integration testing, 20-25. *See also* unit testing

- Monkey, 21-23
- monkeyrunner, 20-21
- UI Automation Viewer, 23-25

intended audience for this book, xiv**Intent filters, 46-47****Intents, 45-48**

- ACTION_NDEF_DISCOVERED intent, filtering, 177-178
- ACTION_TAG_DISCOVERED intent, filtering, 179-181
- ACTION_TECH_DISCOVERED intent, filtering, 178-179
- broadcast receivers, 47-48
- explicit Intents, 45
 - creating, 46
- implicit Intents, 46
 - creating, 46

interaction stage (Bluetooth), 168**Internet, accessing, 127-130****INTERNET permission, 127****interpolators, 118**

J

Java, similarity to Android, xiv**java folder, 36-37****JDKs (Java Development Kits), 3****JetBrains IntelliJ IDEA, 1**

- website, 9

JRE (Java Runtime Environment), 3**JSONArrayRequests, 137****JsonObjectRequests, 137****JUnit**

- adding support for in gradle.build file, 18
- annotations, 19

K-L

KeyboardView view, 60-61**launching AVD Manager, 11****layout folder, 39**

layouts, 71-74

- Android Wear, 218-219
- controlling, 71
- coordinates, 73-74
- frame layout, 81-83
 - overlays, 81-82
 - WebView view, 82-83
- linear layout, 74-77
- LinearLayout, 254
- location reporting application, 147-148
- measurements, 72-73
 - dp, 72-73
 - size groupings, 73
 - sp, 73
- relative layout, 77-79
- table layout, 79-81
- transitioning between, 123-125
- widgets, 86-89
 - home screen, 92-93
 - sizing, 89-90
 - XML layout file, 86-89

Leanback, 252**libraries**

- Leanback, 252
- Volley, 135-137
 - queues, creating, 135-136
 - requests, 136-137

licensing

- EULA, 268
- GenyMotion, 13

lifecycle

- of Activities, 49-51, 101-102
- of Fragments, 53-55

linear layout, 74-77**LinearLayout, 254****ListView, 55****Loaders, 56-57****loading saved games (Google Play), 197-199****Location API, 148-153****location data**

- Fused Location Provider, 148
- Location API, 148-153
- permissions
 - coarse location data, 139-140
 - fine location data, 140
- reporting application, creating, 140-148
 - Activity, 140-144
 - layouts, 147-148
 - services, 144-146

lock screen, widgets

- adding, 92
- layouts, 92-93

Log class, 29**log levels, setting in LogCat, 30-31****LogCat, 29-32**

- log levels, setting, 30-31
- options, 29-30

logging, application logging, 241-242

M

manifest file

- attributes, 35
- declaration, 34
- elements, 35-36
- Intent filters, 46-47
- OpenGL ES, adding as feature, 114-115
- permissions, 127
- updating for app widgets, 96

manifests folder, 34-36

- Activities, adding, 49

measurements for layouts

- dp, 72-73
- size groupings, 73
- sp, 73

MediaRouteButton view, 62**memory**

- available memory, displaying, 27
- images, storing, 107-108
- optimizing, 243-247
 - IntentServices, 244
 - minimizing garbage collection, 244
 - Proguard, 244-245
- profiling, 25-27
- usage, monitoring, 245-247

menu folder, 39**Mercurial, 15****messaging, 29-32**

- LogCat
 - log levels, setting, 30-31
 - options, 29-30

methods

- AppWidgetProvider class, 95-96
- cancelDiscovery(), 169
- Fragments, 52-53
- getActivity(), 55
- getLoaderManager(), 56
- hasResolution(), 204
- KeyboardView, 61
- onCreateLoader(), 57
- onLoaderReset(), 57
- onLoadFinished(), 57
- onPause(), 47-48
- onResume(), 47-48
- onUpdate(), 90
- registerReceiver(), 47-48

- setPriority(), 152
- startDiscovery(), 169
- static methods, 248

MIFARE Classic tags, 177**migrating from ADT to Android Studio, 3-4****minimizing garbage collection, 244****minimum requirements for Android development, xv****mobile data connectivity, detecting, 128****monitoring**

- garbage collection, 245
- memory usage, 245-247

Monkey, integration testing, 21-23**monkeyrunner, 20-21****multimedia**

- audio
 - playback, 156-159
 - recording, 159-161
 - supported codecs, 155
- video, 161-165
 - playback, 162-165
 - supported codecs, 161

MVC (Model-View-Controller)**architecture, 99**

- asynchronous processing, 104-106
 - AsyncTask, 105-106
 - worker threads, 104-105
- content providers, 100-101
 - accessing, 100
 - deleting data from, 101
 - inserting data, 100
 - updating, 100
- Controller, 102-104
- services, 102-104
 - bound state, 103
 - started state, 102
- views, 101-102

N

navigating X/Y axis with remote (Android TV), 253

NDEF (NFC Data Exchange Format) messages, 176

NDK (Native Development Kit), 117

Nearby API, 209-214

Nearby Messages API

- enabling, 209-210
- sending and receiving messages, 210-214

nested XML tags, parsing, 132-133

network detection, 127-128

- mobile data connectivity, detecting, 128
- Wi-Fi connectivity, detecting, 128

networking, 127

- AsyncTask, 133-135
- HTTP clients, 129-130
- Volley, 135-137
 - queues, creating, 135-136
 - requests, 136-137

NFC (Near Field Communication), 176-181

- ACTION_NDEF_DISCOVERED intent, filtering, 177-178
- ACTION_TAG_DISCOVERED intent, filtering, 179-181
- ACTION_TECH_DISCOVERED intent, filtering, 178-179
- NDEF messages, 176
- permissions, 177
- tags, 176-177

NinePatch, 109-111

- Draw 9-patch utility, 110-111

notifications (Android Wear), 224-226

O

ObjectAnimator subclass, 120-121

Official Android Development Site, xv

onCreateLoader() method, 57

onLoaderReset() method, 57

onLoadFinished() method, 57

onPause() method, 47-48

onResume() method, 47-48

onUpdate() method, 90

OpenGL ES, 114-117

- documentation, 117
- GLSurfaceView, setting up, 115-117
- NDK, 117
- texture compression, 115

operating systems, installing Android Studio on OS X, 2-3

optimizing

- applications
 - Application object, extending, 239-241
 - logging, 241-242
 - versioning your configuration, 242-243
- memory management, 243-247
 - IntentServices, 244
 - minimizing garbage collection, 244
 - Proguard, 244-245
- performance, 247-249
 - containers, 249
 - layouts, 248-249
 - objects, 247-248
 - static methods, 248

Oracle VM Virtual Box, 12

OS X operating system, installing Android Studio on, 2-3

overlays, 81-82

overscan, 254

P

parsing XML, 131-133

- ignoring namespaces, 131
- nested tags, 132-133
- text values, retrieving, 131-132

patterns, xiv

performance

- optimizing, 247-249
 - containers, 249
 - for loops, 248-249
 - objects, 247-248
 - static methods, 248
- profiling, 25-27
 - heap dumps, capturing, 26-27
- tracing, 27-28

permissions

- location data
 - coarse location data, 139-140
 - fine location data, 140
- manifest file, 127
- NFC, 177
- verifying, 269

pixels

- bitmaps, 107-111
 - NinePatch, 109-111
- dp, 72-73

playback

- audio, 156-159
- video, 162-165

preferences, backing up, 190-191

preparing for application deployment, 265

Pressed state, 253

Preview pane (Android Studio), 8

previewImage property, 90-92

primitive shapes, 111-113

production checklist for application deployment, 266-270

- certificate keys, 266
- contact email, 266
- external services, 267
- launcher icon, 267-268
- package name, 268-269

profiling, 25-27

ProgressBar view, 62-64

projects

- Android Studio
 - closing, 9
 - starting, 4-9
 - testing folders, creating, 18
- file structure
 - java folder, 36-37
 - manifests folder, 34-36
 - res folder, 37-41
- Gradle build system, 41-42

property animation, 118-121

- AnimatorSet subclass, 121
- defining in XML, 119
- ObjectAnimator subclass, 120-121
- ValueAnimator subclass, 120

publishing applications

- APK generation, 273-274
- Google Play Store
 - Android TV banners, 272
 - charging for your app, 272-273
 - feature graphic, 272
 - high-res icon, 271-272
 - promo graphic, 272
 - promo video, 271
 - screenshots, 271
- preparing for, 265

- production checklist, 266-270
 - certificate keys, 266
 - contact email, 266
 - external services, 267
 - launcher icon, 267-268
 - licensing, 268
 - package name, 268-269
 - removal of excess unused assets, 270
 - verifying permissions, 269

Python scripts, integration testing with monkeyrunner, 20-21

Q-R

queues, creating with Volley, 135-136

random events, throwing with Monkey, 21-23

reading

- device sensor data, 183-185
- files from Google Drive Android API, 193-194
- sensor data, 183-185

Recent Projects list (Android Studio), 4

recommendations for Android TV, 257-258

recording audio, 159-161

RecyclerView view, 67

registering

- applications
 - with Android Backup Services, 189
 - with Developers Console, 191
- this book at InformIT, xvi

registerReceiver() method, 47-48

relative layout, 77-79

requests (Volley), 136-137

res folder, 37-41

- drawable subfolder, 37-38
- menu subfolder, 39

resizing

- images, 108-109
- widgets, 93

retrieving user accounts from devices, 187-188

Robotium automation framework, 20

rotate effect (animation), 118

rows, table layout, 79-81

S

saved games, loading in Google Play Games Services, 197-199

scale effect (animation), 118

scaling

- AVD, 262
- images, 108-109
- virtual devices, 11

scripts, integration testing with monkeyrunner, 20-21

SDKs

- Android SDK, downloading, 9
- Android SDK Manager, 10

Selenium WebDriver, 20

sensors, 181-185

- detecting, 182-183
- Google Fit, 207-209
- reading data, 183-185

services, 102-104

- bound state, 103
- location reporting application, 144-146
- started state, 102

services available in Google Play Services, 202

setPriority() method, 152

shapes, 111-113

singletons, 137

size groupings for layouts, 73
sizing
 images, 108-109
 widgets, 89-90
Snapshots, 197
sound packs, 159
SoundPool class, 156-159
sp (scale-independent pixels), 73
Space view, 64
StackOverflow website, xv
stages of Bluetooth communication, 167-168
startDiscovery() method, 169
started state (services), 102
starting
 Android Studio projects, 4-9
 Activity-selection screen (Android Studio), 6
 audio playback, 161
 AVD Manager, 11
static methods, 248
statistics available with Google Analytics, 232
storing
 code
 Git, 14-15
 Mercurial, 15
 Subversion, 14
 user data, 187-191
styles for layouts
 frame layout, 81-83
 linear layout, 74-77
 relative layout, 77-79
 table layout, 79-81
Subversion, 14
 features, 14

SurfaceView view, 64-65
SVN. See Subversion
system drawables, 113-114
Systrace, 27-28

T

table layout, 79-81
tags (NFC), 176-177
ten-foot view, 252-254
test classes
 annotations, 19
 writing, 18-19
testing, 17
 integration testing, 20-25
 Monkey, 21-23
 monkeyrunner, 20-21
 UI Automation Viewer, 23-25
 unit testing, 17-20
 Appium, 20
 modules, 17
 Robotium automation framework, 20
 test classes, writing, 18-19
text, guidelines for Android TV, 255-256
text values, retrieving from XML, 131-132
texture compression, 115
TextureView view, 65-66
TextView view, 65
threads
 UI, 104
 worker threads, 104-105
throwing random events with Monkey, 21-23
Torvalds, Linus, 14
tracing, 27-28

transition framework, 123-125

defining in XML, 123-124

fade out/in transitions, 125

transitions, 117

translate effect (animation), 118

troubleshooting Android Studio build problems, 6-7

TVs

aspect ratio, 254

functionality, 255

overscan, 254

U

UI

layouts, 71-74

controlling, 71

coordinates, 73-74

measurements, 72-73

lock screen

widgets, adding, 92

threads, 104

views, 59-68

AnalogClock view, 60

buttons, 59

ImageView, 60

KeyboardView, 60-61

MediaRouteButton, 62

ProgressBar, 62-64

Space, 64

SurfaceView, 64-65

TextureView, 65-66

TextView, 65

ViewGroup, 66-67

ViewStub, 68

UI Automation Viewer, 23-25

UI/Application Exerciser Monkey

Monkey, integration testing, 21-23

unit testing

Appium, 20

Robotium automation framework, 20

test classes

annotations, 19

writing, 18-19

testing folders, creating for your project, 18

unregisterReceiver() method, 47-48

update frequency for widgets, 90

updating

content providers, 100

location data, 152

manifest file

app widgets, 96

URI (Uniform Resource Identifier), 100

user accounts

Android Backup Services

Backup Service Key, 189-190

preferences, backing up, 190-191

backing up, 188-191

retrieving from devices, 187-188

utilities

Draw 9-patch, 110-111

Widget Preview, 90-92

zipalign, 244-245

UUID (Universally Unique Identifier), 171

V

ValueAnimator subclass, 120

values folder, 40-41

Verbose log level, 30

verifying permissions, 269

version-control systems

- Git, 14-15
- Mercurial, 15
- Subversion, 14

video

- playback, 162-165
- supported codecs, 161
- YouTube, Android developers page, xv

view animation, 117-118

- alpha, 118
- creating, 118
- interpolators, 118
- rotate effect, 118
- scale effect, 118
- translate effect, 118

ViewGroup view, 66-67

- CardView view, 66-67
- RecyclerView view, 67

views, 59-68

- AnalogClock, 60
- buttons, 59
- custom views, creating, 68-70
- GLSurfaceView, setting up, 115-117
- ImageView, 60
- KeyboardView, 60-61
- Listview, 55
- MediaRouteButton, 62
- MVC architecture, 101-102
- overlays, 81-82
- ProgressBar, 62-64
- Space, 64
- SurfaceView, 64-65
- TextureView, 65-66
- TextView, 65
- transition framework, 123-125

transitions, 117

ViewGroup, 66-67

CardView, 66-67

RecyclerView, 67

ViewStub, 68

WebView, 82-83

ViewStub view, 68

virtual devices

- heap dumps, capturing, 26-27
- profiling, displaying available memory, 27
- scaling, 11

Volley, 135-137

- queues, creating, 135-136
- requests, 136-137

W

Warn log level, 30

WatchViewStub class, 218-219

Wearable UI Library, 218

web resources for Android TV, 253

websites

- Android development, xv
- Android Studio, xv, 2
- Git, 14
- Gradle, 42
- JetBrains IntelliJ IDEA, 9
- Robotium automation framework, 20
- Subversion, 14

WebView view, 82-83

Welcome Screen (Android Studio), 4

Widget Preview utility, 90-92

widgets, 85-86

- adding to lock screen, 92
- Android TV, 258
- AppWidgetProvider class, 94-96

- AppWidgetProviderInfo object, 89-93
- creating, 85-86
- layouts, 86-89
 - home screen, 92-93
 - sizing, 89-90
 - XML layout file, 86-89
- manifest file entries, 96
- preview image, 90-92
- resizing, 93
- update frequency, 90
- Wi-Fi connectivity, detecting, 128**
- worker threads, 104-105**
- writing**
 - to files with Google Drive API, 194-195
 - test classes, 18-19
- WTF (Assert, What a Terrible Failure) tag, 32**

X

- Xamarin Android Player, 13**
- XML**
 - custom views, 69
 - drawable animation, defining, 122
 - layout folder, 39
 - layouts, 71
 - measurements, 72-73
 - widget layout files, 86-89
 - manifest file, 34-36
 - attributes, 35
 - declaration, 34
 - elements, 35-36
 - Intent filters, 46-47
 - OpenGL ES, adding as feature, 114-115
 - permissions, 127
 - updating for app widgets, 96

- parsing, 131-133
 - ignoring namespaces, 131
 - nested tags, 132-133
 - text values, retrieving, 131-132
- property animations, defining, 119
- sample animation XML, 118
- shapes, defining, 112-113
- transition frameworks, defining, 123-124
- values folder, 40-41

XmlPullParser, 131**X/Y axis**

- navigating with remotes (Android TV), 253
- obtaining layout coordinates, 73-74
- translate effect (animation), 118

Y-Z

YouTube, Android developers page, xv

- z-index, frame layout, 81-83**
- zipalign tool, 244-245**