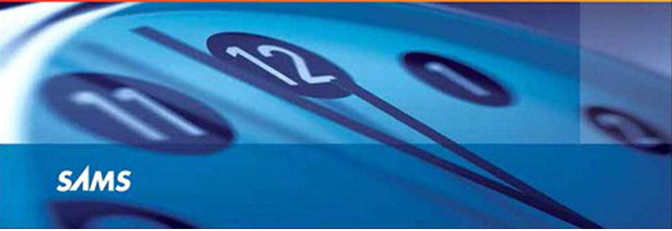
Carmen Delessio
Lauren Darcey
Shane Conder

In **Full Color**

Fourth Edition

Features Android Release 5.0

Sams **Teach Yourself** Android™ Application Development in 24 Hours



SAMS

FREE SAMPLE CHAPTER

Praise for *Sams Teach Yourself Android™ Application Development in 24 Hours*, Fourth Edition

"This latest edition of *Sams Teach Yourself Android Application Development in 24 Hour*s is just what you've been waiting for if you've been waiting to get into Android development. Freshly updated with what you need to know for developing applications using Android Studio for Android Lollipop (Android 5) with Material Design, this book covers what you need to know to get started building applications for Android."

—**Ray Rischpater**, Author and Engineering Manager at Microsoft

"The new edition of *Sams Teach Yourself Android Application Development in 24 Hours* covers a lot of new features. The book takes you from the beginning through to uploading your own app into the store. All the screen shots in this edition use the new and official Android IDE (the amazing Android Studio IDE)."

—**Fady A. M. Ibrahim**, Android Instructor, Benha Faculty of Computer and Information

"Any developer who wants to get up to speed quickly on Android will appreciate this introduction. Beyond the SDK fundamentals, there's plenty of good information on the things real-world Android apps are made of, such as maps, images, and navigation. This is a great way to dive head-first into Android development, or just to become Android-literate in record time."

—**Jonathan Taylor**, VP, Mobile Technology, Priceline.com

The authors knock it out of the park for new Android developers and experienced ones who want to extend their prowess. This book is perfectly set-up for a sports technology oriented person like me to teach me the basic principles, give me design knowledge, and then cap that off with how to add and manipulate data. Data-driven applications are the life's blood of every fantasy sports player and the authors' ability to break down the path to success with real-life exercises to put these principles into action is a Grand Slam!"

—**Rick Wolf**, President, Fantasy Alarm, and Co-Founder, Fantasy Sports Trade Association

This page intentionally left blank

Carmen Delessio
Lauren Darcey
Shane Conder

Sams Teach Yourself Android™ Application Development

Fourth Edition

in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Android™ Application Development in 24 Hours, Fourth Edition

ISBN-13: 978-0-672-33739-0

ISBN-10: 0-672-33739-8

Library of Congress Control Number: 2015906279

Printed in the United States of America

First Printing July 2015

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

Special Sales

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

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

For questions about sales outside the U.S., please contact international@pearsoned.com.

Acquisitions Editor
Laura Lewin

Development Editor
Sheri Cain

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Keith Cline

Indexer
Larry Sweazy

Proofreader
Sarah Kearns

Technical Editors
Ray Rischpater
Valerie Shipbaugh

Publishing Coordinator
Olivia Basegio

Interior Designer
Gary Adair

Cover Designer
Mark Shirar

Composition
Nonie Ratcliff

Contents at a Glance

Preface xiv

Part I Android Fundamentals

HOUR 1 Introducing Android 3
2 Understanding Intents 17
3 Understanding Resources 37
4 Activities and Fragments 53
5 Responsive Apps: Running in the Background 73

Part II Creating the User Interface

HOUR 6 Using Basic UI Controls 91
7 Using Layouts 105
8 ListViews and Adapters 119
9 Material Design 139
10 More Views and Controls ..... `161
11 ImageViews and Bitmaps 179
12 Using VideoViews and Media 195
13 Adding Navigation 207

Part III Working with Data

HOUR 14 Using the File System 233
15 Using SharedPreferences 245
16 Using SQLite and File Storage 261
17 Accessing the Cloud: Working with a Remote API 277
18 Introducing Content Providers 297
19 Creating a Content Provider 311
20 Loaders and CursorAdapters 323

Part IV Next Steps

HOUR 21 Using Notifications 337
22 Android TV and Wear Apps 347
23 More Features to Explore 361
24 Publishing Your Apps 375
Index 387

Table of Contents

Preface xiv

Part I: Android Fundamentals **1**

HOUR 1: Introducing Android **3**

Defining Android 3

Understanding Android Development 5

Beginning Android Studio 10

Summary 14

Q&A 14

Workshop 14

Exercise 15

HOUR 2: Understanding Intents **17**

Using Intents to Start an Activity 17

Using Implicit Intents 27

Handling an Implicit Intent 32

Summary 34

Q&A 35

Workshop 35

Exercise 36

HOUR 3: Understanding Resources **37**

Resources in Your Project 37

Understanding Common Resources 40

Providing Alternative Resources 48

Internationalization: Using Alternative Languages Resources 49

Summary 51

Q&A 52

Workshop 52

Exercises 52

HOUR 4: Activities and Fragments 53
- Working with Activities 53
- Understanding the Activity Lifecycle 63
- Introducing Fragments 66
- Summary 71
- Q&A 71
- Workshop 71
- Exercises 72

HOUR 5: Responsive Apps: Running in the Background 73
- Working in the Background 73
- Using an AsyncTask 76
- Service and IntentService 79
- Summary 87
- Q&A 87
- Workshop 88
- Exercise 88

Part II: Creating the User Interface 89

HOUR 6: Using Basic UI Controls 91
- Using the Android Studio Palette 91
- Handling User Input 92
- Using Buttons for User Actions 97
- Summary 102
- Q&A 103
- Workshop 103
- Exercise 103

HOUR 7: Using Layouts 105
- Getting Started with Layouts 105
- Becoming a RelativeLayout Expert 108
- Common Attributes 112
- More Layout Types 114
- Summary 116

Q&A 117
Workshop 117
Exercises 117

HOUR 8: ListViews and Adapters 119
Getting Started with ListViews 119
Extending Base Adapters 124
Introducing the View Holder Pattern 131
Summary 136
Q&A 136
Workshop 136
Exercises 137

HOUR 9: Material Design 139
The Evolution of App Design 139
Introducing Material Design 140
Implementing Material Design 145
Summary 159
Q&A 159
Workshop 160
Exercise 160

HOUR 10: More Views and Controls 161
Controls for Collecting Information 161
Indicating Progress 166
Displaying Data 170
More Views 174
Summary 177
Q&A 177
Workshop 177
Exercise 178

HOUR 11: ImageViews and Bitmaps 179
Examining ImageView 179
Using Bitmaps and Canvas 186
Introducing Picasso 192

Summary 192
Q&A 193
Workshop 193
Exercise 193

HOUR 12: Using VideoViews and Media 195
Playing Video 195
Handling VideoView Events 199
Playing Audio with MediaPlayer 202
More Media 204
Summary 205
Q&A 205
Workshop 205
Exercise 206

HOUR 13: Adding Navigation 207
Using the ActionBar 207
Introducing the Toolbar 214
Sliding Drawer Navigation 219
Summary 228
Q&A 228
Workshop 229
Exercise 229

Part III: Working with Data 231

HOUR 14: Using the File System 233
File System Overview 233
Saving Data Privately in Your App 236
Saving Data in Public Folders 240
Summary 242
Q&A 242
Workshop 242
Exercise 243

HOUR 15: Using SharedPreferences 245
Using SharedPreferences to Store Data 245
Setting User Preferences 249
Summary 258
Q&A 258
Workshop 258
Exercise 259

HOUR 16: Using SQLite and File Storage 261
Organizing a Database with Tables 261
Managing Data with SQLiteOpenHelper 263
Adding, Deleting, and Updating Data 266
Querying Data and Using Cursors 267
Using a Database in the App 269
Summary 274
Q&A 274
Workshop 274
Exercise 275

HOUR 17: Accessing the Cloud: Working with a Remote API 277
Fetching Remote Data 277
Using and Parsing JSON-Formatted Data 282
Putting the Pieces Together 286
Checking Connectivity 290
Summary 294
Q&A 295
Workshop 295
Exercise 295

HOUR 18: Introducing Content Providers 297
Introducing Content Providers 297
All About the Calendar 298
Calendar Data via the Calendar Content Provider 300
Summary 308

Q&A 308
Workshop 309
Exercise 309

HOUR 19: Creating a Content Provider 311
Specifying a URI for Data Retrieval 311
Using PieDbAdapter 312
Building a Content Provider 312
Using MyContentProvider in the App 319
Summary 321
Q&A 321
Workshop 322
Exercise 322

HOUR 20: Loaders and CursorAdapters 323
How Loaders Work 323
Loader Classes 324
Understanding Loader States 325
Creating Cursor Adapters 330
The Rest of the App 333
Summary 333
Q&A 333
Workshop 334
Exercise 334

Part IV: Next Steps 335

HOUR 21: Using Notifications 337
Introducing Notifications 337
Creating and Managing Notifications 338
Customizing Notifications 344
Summary 345
Q&A 346
Workshop 346
Exercise 346

HOUR 22: Android TV and Wear Apps 347
Android as a Platform 347
Developing Android Wear Apps 348
Developing Android TV Apps 355
Summary 358
Q&A 358
Workshop 359
Exercise 359

HOUR 23: More Features to Explore 361
Using Google Play Services 361
Using Google Play Services for Location 364
Using Open Source and External SDKs 368
Digging Deeper into Android 369
Summary 373
Q&A 373
Workshop 374
Exercise 374

HOUR 24: Publishing Your Apps 375
Preparing for Release 375
Sharing Your App with the World 382
Monetizing Your App 384
Summary 385
Q&A 385
Workshop 385
Exercise 386

Index 387

Preface

What I wish I knew when I started Android development....

Android has become a leading platform for smartphones, tablets, and other devices. The goal of this book is to introduce the Android platform and start you on the path to creating professional-grade apps.

In 24 hours of topic-based material, you learn the concepts of Android development and move on to specific topics like working with data in the cloud, handling bitmaps and videos in an app, and using new features (such as `CardView` in the Lollipop versions of Android). The coverage of material design and Lollipop features will take you far. The Lollipop version of Android will give way to Android M in the future. Android M will focus on performance improvements.

In the early days of Android, author Carmen Delessio worked on a significant Android project for a large media company. The app launched and was a success. But, it could have been built in a more "Android way." With the authors having built many Android apps since then, the material in this book is largely guided by the idea of including "what I wish I knew then."

This book is not intended to be an encyclopedia of all things Android. Plenty of Android resources are available, and the documentation on the Android developer site has never been better. This book starts you on the path to developing professional Android apps and can be used as a guide to the additional material.

New in the Fourth Edition

There are two major changes from the third edition to the fourth edition of this book.

New features in Android are covered. The updates include significant coverage of material design, including `RecyclerView` and `CardView`. New notification features are covered. An introduction to Android Wear and Android TV is included. Significantly, this edition uses Android Studio throughout rather than Eclipse. All development screenshots and examples use Android Studio.

The second change is this book starts by covering four important components of Android. In the first hour, you learn about activities, intents, intent services, and broadcast receivers. Subsequent hours drill more deeply into these broad concepts. This change highlights what is happening when an app runs and puts even more emphasis on doing things the "Android way."

Who This Book Is For

The examples in this book are created so that someone with programming knowledge can understand them, but Android apps are developed in Java. You will find the book much more valuable and useful if you are familiar with Java concepts and syntax. If you are knowledgeable in C or C# and understand object-oriented concepts, you should be able to understand the level of Java code in this book. You should know what classes and methods are.

If you are a Java programmer with an interest in Android development, this book introduces you to Android and gets you on track for professional Android development.

If you have started Android development, but have not proceeded past the basic examples, this book is for you. It covers topics such as downloading data, using a database, and creating content providers. This book can take you from the basics to real development in a series of understandable steps.

How This Book Is Organized

The book is organized into four broad sections:

Part I, "Android Fundamentals." This first part introduces Android concepts and uses examples to show how to start activities, pass data, and handle core functionality. It covers activities, intents, resources, and background processing.

Part II, "Creating the User Interface." When you create the user interface, you learn about components and layouts. You cover bitmaps and video views. You also learn about navigation within an app, and you cover material design—the new design from Google that is used in Android.

Part III, "Working with Data." Working with data means both retrieving data over a network and storing it. You learn about using a SQLite database and using content providers.

Part IV, "Next Steps." This last part covers other features to investigate further, open source projects of interest, and how to publish your app.

You can find online updates, contact the author, and ask questions about this book on http://talkingandroid.com/. Links to source code are posted there.

Source Code for the Book

Nearly every chapter in this book includes an example that has source code available online. The code is on GitHub and organized by chapter. You will find the code here: https://github.com/CarmenDelessio. Code for an individual chapter should be easy to find. For example, the complete project code for Hour 10 is here: https://github.com/CarmenDelessio/Hour10application.

About the Authors

Carmen Delessio is an experienced application developer who has worked as a developer, technical architect, and CTO in large and small organizations. Carmen began his online development career at Prodigy, where he worked on early Internet applications, shopping apps, and fantasy baseball. He is a graduate of Manhattanville College and lives in Pound Ridge, New York, with his wife, Amy, and daughter, Natalie.

Lauren Darcey is responsible for the technical leadership and direction of a small software company specializing in mobile technologies, including Android and iOS consulting services. With more than two decades of experience in professional software production, Lauren is a recognized authority in application architecture and the development of commercial-grade mobile applications. Lauren received a BS in computer science from the University of California, Santa Cruz.

Shane Conder has extensive application development experience and has focused his attention on mobile and embedded development for well over a decade. He has designed and developed many commercial applications for Android, iOS, BREW, BlackBerry, J2ME, Palm, and Windows Mobile—some of which have been installed on millions of phones worldwide. Shane has written extensively about the tech industry and is known for his keen insights regarding mobile development platform trends. Shane received a BS in computer science from the University of California, Santa Cruz.

Dedication

For ASL and NMLD.

"To the Valiant of heart, nothing is impossible." – Jeanne d'Albret
—Carmen Delessio

Acknowledgments

This book would not exist without the help and guidance of the team at Pearson (Sams Publishing). Thanks to Laura Lewin for constant encouragement and Olivia Basegio for her incredible work on the project. Sheri Cain helped take this book to another level with her feedback. Her diligence and hard work kept this project constantly moving forward.

Technical editors are an important part of every book. Ray Rischpater was an incredible help. Valerie Shipbaugh did her technical review by placing herself in the role of a reader who was new to Android. The feedback and guidance from Ray and Valerie make this a better book.

We Want to Hear from You

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@samspublishing.com

Mail: Reader Feedback
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at http://www.informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

HOUR 11
ImageViews and Bitmaps

What You'll Learn in This Hour:

- Examining `ImageView` Bitmaps
- Using `Bitmaps` and `Canvas`
- Introducing Picasso

Images and media can play an important role in creating an exceptional Android app. In this chapter, you look at the details of handling images and bitmaps, including creating bitmaps, using drawing commands, and handling very large images.

Examining ImageView

You learned about different types of views in Hour 10, "More Views and Controls." An `ImageView` is a view that displays an image, but you will find that there are unique aspects to working with images. An `ImageView` can display any drawable image. The source of the image can be a resource, a drawable, or a bitmap.

Displaying an Image

There are four methods available for setting an image in an `ImageView`. They differ by how the image to display is defined. The image can be a `bitmap`, `drawable`, `Uri`, or resource `id`. The methods are as follows:

- **`setImageDrawable()`**: Set a drawable as the content of the `ImageView`.
- **`setImageBitmap()`**: Set a `Bitmap` as the content of the `ImageView`.
- **`setImageResource()`**: Use a resource `id` to set the content of the `ImageView`.
- **`setImageUri()`**: Use a URI to set the content of the `ImageView`.

To set an `ImageView` to an image resource defined by `R.drawable.mainImage`, you use the following:

```
ImageView mainImage = (ImageView) findViewById(R.id.imageView1);
mainImage.setImageResource(R.drawable.mainImage)
```

To populate a `Drawable` object from a resource, use the `getResources.getDrawable()` method:

```
Drawable myDrawable = getResources().getDrawable(R.drawable.ic_launcher);
```

In this hour, you populate an `ImageView` using a resource `id` as the source and then explore several properties of how an `ImageView` can display an image.

Using ScaleTypes in ImageView

`ImageViews` include a `ScaleType` property. The `ScaleType` defines how the image will be displayed within the `ImageView`. Using `ScaleType`, you can have an image fill the entire `ImageView`, be centered in the `ImageView`, or be cropped and centered in the `ImageView`.

The options for `ScaleType` are defined in `ImageView.ScaleType`. For example, `ImageView.ScaleType.CENTER` refers to a scale type in which the image is centered in the `ImageView`. The complete set of `ScaleTypes` are as follows:

- **`ImageView.ScaleType.CENTER`**: Center the image with no scaling. The image dimensions are unchanged.
- **`ImageView.ScaleType.CENTER_CROP`**: Scales the image and keeps the aspect ratio until either the width of height of the image is the same as the width or height of the `ImageView`. For a small image, this has the effect of enlarging the entire image. For a large image, this has the effect of showing the center of the image.
- **`ImageView.ScaleType.CENTER_INSIDE`**: The image is scaled, and the aspect ratio is maintained. The width and height of the image fit within the `ImageView`.
- **`ImageView.ScaleType.FIT_CENTER`**: Maintain aspect ratio and fit the image in the center of the `ImageView`.
- **`ImageView.ScaleType.FIT_START`**: Maintain aspect ratio and fit the image in the left and top edge of the `ImageView`.
- **`ImageView.ScaleType.FIT_END`**: Maintain aspect ratio and fit the image in the right and bottom edge of the `ImageView`.
- **`ImageView.ScaleType.FIT_END`**: Maintain aspect ratio and fit the image in the right and bottom edge of the `ImageView`.
- **`ImageView.ScaleType.MATRIX`**: Scale using a matrix.

You can change scaleType dynamically in your code. Listing 11.1 show the code for an app that displays an ImageView and includes a RadioGroup and set of RadioButtons for changing the scale type. When a radio button is selected, the scaleType for the ImageView is updated.

LISTING 11.1 Changing ScaleType Programatically

```
package com.talkingandroid.hour11application;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.RadioGroup;

public class ScaleActivity extends Activity {
    RadioGroup radioGroup;
    ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_scale);
        radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
        imageView = (ImageView) findViewById(R.id.imageView);
        radioGroup.setOnCheckedChangeListener(new
                         RadioGroup.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId) {
                switch (checkedId){
                  case R.id.radioCenter:
                    imageView.setScaleType(ImageView.ScaleType.CENTER);
                    break;
                  case R.id.radioCenterCrop:
                    imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
                    break;
                  case R.id.radioCenterInside:
                    imageView.setScaleType(ImageView.ScaleType.CENTER_INSIDE);
                    break;
                  case R.id.radioFitCenter:
                    imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
                    break;
                  case R.id.radioFitStart:
                    imageView.setScaleType(ImageView.ScaleType.FIT_START);
                    break;
                  case R.id.radioFitEnd:
                    imageView.setScaleType(ImageView.ScaleType.FIT_END);
                    break;
                  case R.id.radioFitXY:
```

```
41:                      imageView.setScaleType(ImageView.ScaleType.FIT_XY);
42:                      break;
43:                }
44:            }
45:        });
46:    }
47:}
```

On line 17 of Listing 11.1, an `OnCheckChangeListener()` is set for the `RadioGroup`. When the change is detected, the select `RadioButton id` is checked, and the appropriate `scaleType` is set on the image.

The image used in the code for Listing 11.1 is shown in Figure 11.1. The image is 900 pixels wide and 200 pixels high. It is used in several other examples in this chapter.

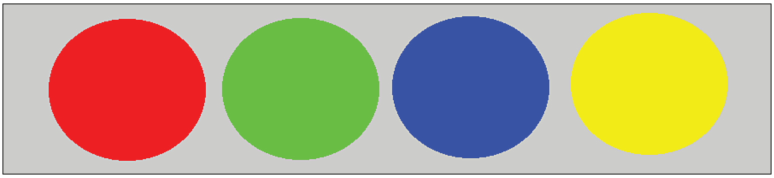


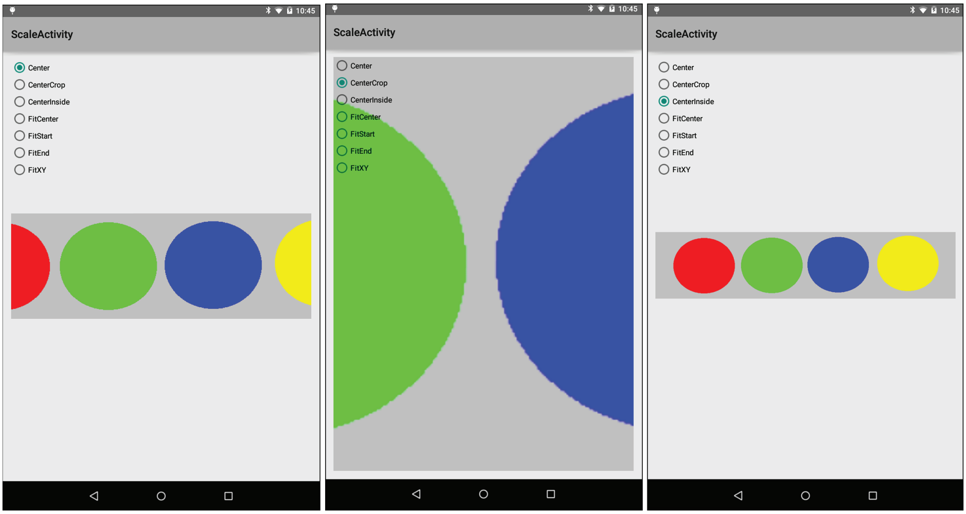
FIGURE 11.1
Base image for showing `ScaleType` (scaletest.png).

By using this simple image with four circles of different colors, it is easy to see the effect of the changing `ScaleType`.

The `ImageView` is set to match the parent width and height. When the image `scaleType` is set to `CENTER_INSIDE`, the image is shown taking the full width of the `ImageView` and is centered with a height that is proportional to the width.

Figure 11.2 shows the base image using the `scaleTypes` set to `CENTER`, `CENTER_CROP`, and `CENTER_INSIDE`. Using `CENTER` shows the image in actual size. Because the size of the image is larger than the `ImageView`, the green and blue circles in the center are shown. `CENTER_CROP` shows half of the green and blue circle. The height of the image fills the `ImageView`. `CENTER_INSIDE` shows the entire image centered in the `ImageView`.

Figure 11.3 shows the base image using the `ScaleTypes FIT_CENTER`, `FIT_START`, `FIT_END`, and `FIT_XY`. The aspect ratio is maintained in the first three, but when using `FIT_XY`, the image fills the `ImageView` and “stretches” the image to fit.



ScaleActivity

● Center
○ CenterCrop
○ CenterInside
○ FitCenter
○ FitStart
○ FitEnd
○ FitXY

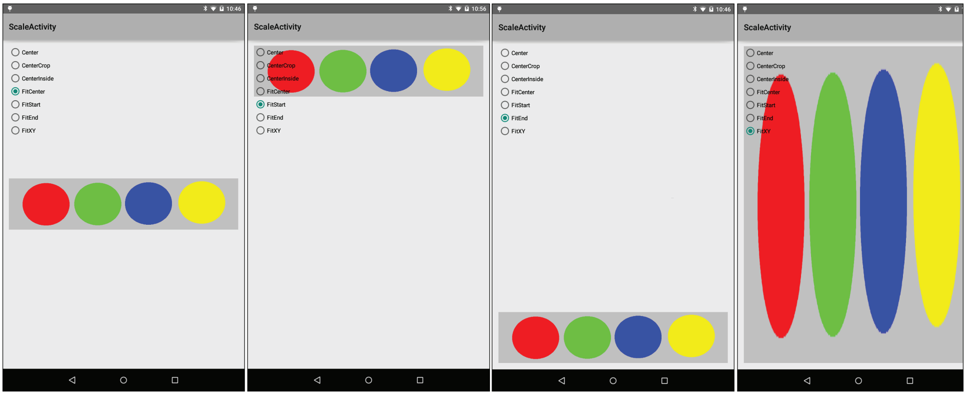
ScaleActivity

○ Center
● CenterCrop
○ CenterInside
○ FitCenter
○ FitStart
○ FitEnd
○ FitXY

ScaleActivity

○ Center
○ CenterCrop
● CenterInside
○ FitCenter
○ FitStart
○ FitEnd
○ FitXY

FIGURE 11.2
`ScaleTypes CENTER`, `CENTER_CROP`, and `CENTER_INSIDE`.



ScaleActivity

○ Center
○ CenterCrop
○ CenterInside
● FitCenter
○ FitStart
○ FitEnd
○ FitXY

ScaleActivity

○ Center
○ CenterCrop
○ CenterInside
○ FitCenter
● FitStart
○ FitEnd
○ FitXY

ScaleActivity

○ Center
○ CenterCrop
○ CenterInside
○ FitCenter
○ FitStart
● FitEnd
○ FitXY

ScaleActivity

○ Center
○ CenterCrop
○ CenterInside
○ FitCenter
○ FitStart
○ FitEnd
● FitXY

FIGURE 11.3
`ScaleTypes FIT_CENTER`, `FIT_START`, `FIT_END`, and `FIT_XY`.

Rotating an Image

An `ImageView` contains several methods for rotating an image. When you rotate an image, you must set the point in the image to rotate around. That is the *pivot point*. The method `setPivotX()` and `setPivotY()` are used to set the pivot point.

Once the pivot point is set, you can call the `setRotation()` method to make the image actually rotate.

The idea in Listing 11.2 is to set the pivot point to the center of the ImageView and to rotate the image 30 degrees each time the button is clicked. The ImageView is defined to have height and width set to match_parent. The ImageView occupies the entire screen.

To get the center of the ImageView, the width and height are divided by 2. To continuously rotate, the number of clicks count is kept. The angle to rotate is 30 times the number of clicks. So, if the button is clicked twice, the image is rotated 60 degrees.

Figure 11.4 shows the rotated image.

LISTING 11.2 Rotating an Image

```
package com.talkingandroid.hour11application;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class RotateActivity extends Activity {
    Button rotateButton;
    ImageView imageView;
    int numClicks = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rotate);
        imageView = (ImageView)findViewById(R.id.imageView);
        rotateButton = (Button) findViewById(R.id.button);
        rotateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                imageView.setPivotX(imageView.getWidth()/2);
                imageView.setPivotY(imageView.getHeight() / 2);
                imageView.setRotation(30*numClicks);
                numClicks++;
            }
        });
    }
}
```

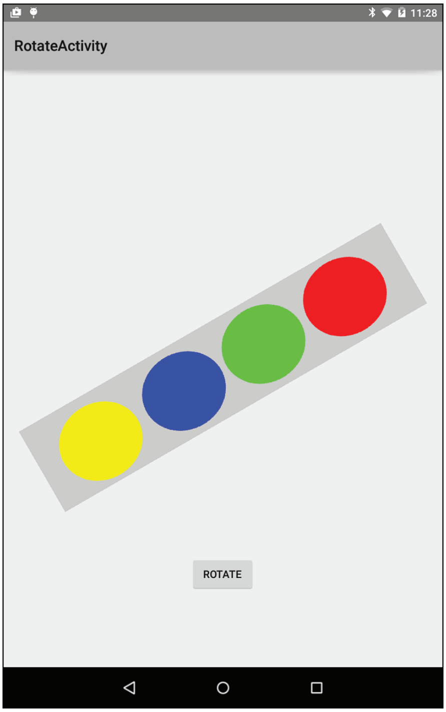


FIGURE 11.4
Rotated image.

NOTE

Considering a Matrix

As an alternative to using the built-in rotation method, you can use a matrix with an `ImageView`. In graphics programming, a matrix is used to transform an image. Simple transformations include scaling, translating, or rotating an image. Android includes a `Matrix` class `(android.graphics.Matrix)` to support these graphic transformations.

Setting Alpha

Alpha level indicates the opacity of an image. An image can be completely transparent, completely opaque, or somewhere in the middle. The alpha level can be set on an `ImageView` using the `setAlpha()` method or, since API level 11, the `setImageAlpha()` method. These methods take an integer parameter. A parameter of 0 indicates complete transparency and 255 for complete opacity.

▼ TRY IT YOURSELF

Using a SeekBar to Dynamically Change Alpha Values

You'll create an Activity with a SeekBar and an ImageView. The range of the SeekBar will be from 0 to 255. As the SeekBar moves, set the alpha value for the ImageView and watch the change:

1. Create a new project with an Activity.
2. Change the XML layout file to include a SeekBar and an ImageView.
3. The ImageView should have the src value set to an existing image.
4. Use setOnSeekBarChangeListener() to set an OnSeekBarChangeListener for the SeekBar.
5. In the OnSeekBarChangeListener, you implement three methods: onProgressChanged(), onStartTrackingTouch(), and onEndTrackingTouch(). If you are using Android Studio, these methods are created for you if you type "new OnSeekBarChangeListener()" as a parameter to the setOnSeekBarChangeListener() method.
6. The onProgressChanged() method includes a parameter called progress that indicates the value on the SeekBar. Use that value to change the alpha level of the ImageView. You will use imageView.setImageAlpha(progress);.

Using Bitmaps and Canvas

The Bitmap(android.graphics.Bitmap) class represents a bitmap image. Bitmaps are created via the BitmapFactory(android.graphics.BitmapFactory) class.

Three typical ways use BitmapFactory to create Bitmaps are to create a bitmap from a resource, file, or InputStream. To create a Bitmap from a resource, use the BitmapFactory method decodeResource():

```
Bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.someImage);
```

The other two methods are similar to decodeResource(): decodeFile() and decodeStream().

Handling Large Images

There are techniques for avoiding the dreaded out-of-memory (OOM) exception. Large images can have a significant impact on memory use in your app. To demonstrate this, you'll create an unrealistically large image to display in an ImageView. If the unmodified image is loaded into an ImageView, the app fails with an OOM error. A java.lang.OutOfMemory exception occurs. You'll fix the memory error for this case by checking the image size and display side.

The idea is to display the image at an appropriate size for the device. There is no point in showing a 10-foot mural in a 6-inch frame. Similarly, there is no point in showing a 20-inch image on a 3-inch device screen. You will scale down the image size and save memory.

The details of your app will influence your memory usage and the techniques that will work best in your case. This example shows how to handle a single large image.

To demonstrate this, you'll start with an image and increase it to an *unrealistic* size. You will use a photo that is 72 inches x 54 inches and that has a 28MB file size.

The image is in the drawable resource folder and has the id R.drawable.largeimage.

You can cause the app to fail with an OOM error by trying to set an ImageView to this resource. You have an ImageView named imageView. This line of code that causes the app to fail is this:

```
imageView.setImageResource(R.drawable.largeimage);
```

Some work is required, but it is possible to handle an image this large. In all cases, it would be better to work with appropriately sized images, but that does not always happen.

The approach is to get the dimensions of the underlying Bitmap without actually rendering it. Getting those dimensions is not a memory-intensive activity. Once you have the Bitmap, you can determine an appropriate size for the Bitmap that will fit in our display. If you have a 20-inch image and a 4-inch display, you'll request that the Bitmap that is created in memory fill the 4-inch display.

Using BitmapFactory.Options

The BitmapFactory.Options class is used with the BitmapFactory class. It is essential for handling large bitmaps.

You'll use the following options from the BitmapFactoryOptions class:

- **inJustDecodeBounds:** If set to true, this option indicates that the Bitmap dimensions should be determined by the BitmapFactory but that the Bitmap itself should not be created. This is the key to getting the Bitmap dimensions without the memory overhead of creating the Bitmap.
- **outWidth:** The width of the image set when inJustDecodeBounds is used.
- **outHeight:** The height of the image set when inJustDecodeBounds is used.
- **inSampleSize:** This integer indicates how much the dimensions of the Bitmap should be reduced. Given an image of 1000x400, an inSampleSize of 4 will result in a Bitmap of 250x100. The dimensions are reduced by a factor of 4.

Listing 11.3 shows the code to address this.

LISTING 11.3 Displaying a Large Image

```
 1: package com.talkingandroid.hour11application;
 2: import android.app.Activity;
 3: import android.graphics.Bitmap;
 4: import android.graphics.BitmapFactory;
 5: import android.os.Bundle;
 6: import android.view.Display;
 7: import android.widget.ImageView;
 8:
 9: public class LargeImageActivity extends Activity {
10:
11:     @Override
12:     protected void onCreate(Bundle savedInstanceState) {
13:         super.onCreate(savedInstanceState);
14:         setContentView(R.layout.activity_large_image);
15:         ImageView imageView = (ImageView) findViewById(R.id.imageView);
16:         Display display = getWindowManager().getDefaultDisplay();
17:         int displayWidth = display.getWidth();
18:         BitmapFactory.Options options = new BitmapFactory.Options();
19:         options.inJustDecodeBounds = true;
20:         BitmapFactory.decodeResource(getResources(), R.drawable.largeimage,
21:              options);
22:         int width = options.outWidth;
23:         if (width > displayWidth) {
24:             int widthRatio = Math.round((float) width / (float) displayWidth);
25:             options.inSampleSize = widthRatio;
26:         }
27:        options.inJustDecodeBounds = false;
28:        Bitmap scaledBitmap =  BitmapFactory.decodeResource(getResources(),
29:                               R.drawable.largeimage, options);
30:        imageView.setImageBitmap(scaledBitmap);
31:    }
32: }
```

On lines 16 and 17, you get the size of the device display. You'll use this as the target size for reducing the image size.

On lines 18–22, you determine the size of the current `Bitmap`. You do that by creating a `BitmapFactory.Options` class and setting the `inJustDecodeBounds` value to `true`. On line 20, the `Bitmap` is decoded to get the dimensions. Using this method, you get the dimensions without the memory overhead of creating the `Bitmap`. The result is available in `options.outWidth`. On line 22, you assign `options.outWidth` to the `int` variable `width`.

In this example, you use a simple test for the size of the image. On line 23, you check whether the width of the `Bitmap` is greater than the size of the display. If that is the case, you must determine the `inSampleSize` to use. That is done on lines 24 and 25. If the width of the `Bitmap` is 1000 pixels and the size of the display is 250 pixels, you get an `inSampleSize` of 4 by dividing the width of the `Bitmap` by the width of the display. For simplicity, you are not checking the height.

With the `imSampleSize` set to an appropriate value, you can render the image.

`On` line 27, the `inJustDecodeBounds` value is set to false. That means the image will be decoded and a `Bitmap` object will be created.

Lines 28 and 29 use the `BitmapFactory.decodeResource()` method to actually decode the image and create the `Bitmap`. The bitmap is assigned to the variable `scaledBitmap`. It is important to note that in this call, the `BitmapFactory.Options` variable `options` is passed as a parameter. That is how you indicate to the `BitmapFactory` what `inSampleSize` to use. The value for `options.inSampleSize` was set on line 25.

It is certainly not recommended to display a 72-inch image on a device, but Figure 11.5 shows that it can be done!

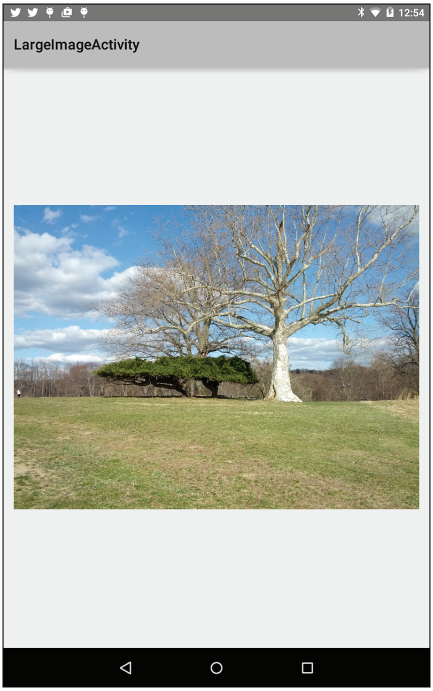


FIGURE 11.5
Very large photo displayed on device.

Drawing Directly on a Canvas

There is one more thing that you can do with an ImageView and Bitmap. You'll create a Bitmap and draw directly on the Canvas that is associated with the Bitmap. A Canvas is an object that you can draw on by calling drawing commands.

You will use an ImageView to display the Bitmap. You will also use an ImageView to determine the dimensions when creating the Bitmap and for drawing. In Listing 11.4, you draw the word "Hello" in the center of the screen.

LISTING 11.4 Drawing on a Canvas

```
package com.talkingandroid.hour11application;
import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class DrawActivity extends Activity {
    ImageView imageView;
    Button drawButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_draw);
        imageView = (ImageView) findViewById(R.id.imageView);
        drawButton = (Button)findViewById(R.id.button);
        drawButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Bitmap imageBitmap = Bitmap.createBitmap(imageView.getWidth(),
                       imageView.getHeight(), Bitmap.Config.ARGB_8888);
                Canvas canvas = new Canvas(imageBitmap);
                float scale = getResources().getDisplayMetrics().density;
                Paint p = new Paint();
                p.setColor(Color.BLUE);
                p.setTextSize(48*scale);
                canvas.drawText("Hello", imageView. getWidth()/2,
                                 imageView.getHeight()/2, p);
                imageView.setImageBitmap(imageBitmap);
            }
        });
    }
}
```

A new Bitmap is created on lines 25 and 26 by using the method Bitmap.createBitmap(). Note that the width and height of the bitmap are set using the width and height of the ImageView. The Bitmap.Config (android.graphics.Bitmap.Config) is set to Bitmap.Config.ARGB_8888.

When looking at the documentation for the Bitmap class, there are a number of createBitmap() methods that take different parameters. These methods may return a *mutable* or an *immutable* Bitmap. That is important; only a mutable Bitmap can be used for drawing.

On line 27, a Canvas is instantiated based on the Bitmap that you created.

Simple drawing commands are applied to the canvas in lines 28–33. You create a Paint object and set the Color to blue and set the text size. Line 28 gets the density of the display. That is used to set the text size properly. Recall that you previously learned about converting density independent pixels to pixels. On line 32, you draw the word "Hello" in the center of the Canvas.

On line 34, you update the ImageView to show your generated Bitmap.

Figure 11.6 shows the result.

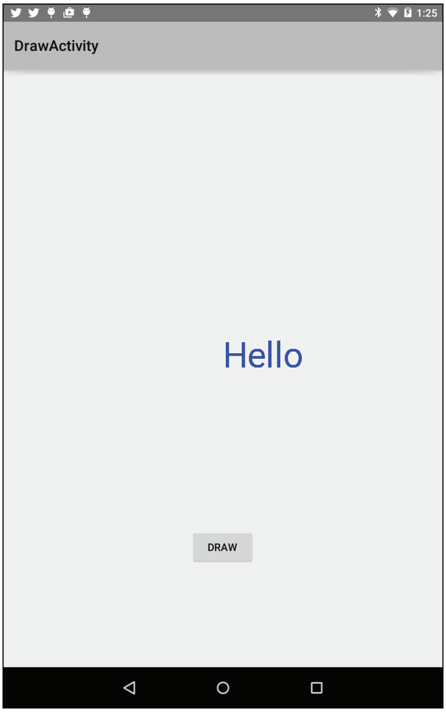


FIGURE 11.6
Drawing on a Canvas.

Introducing Picasso

Picasso is an open source Android library from the team at Square. Picasso is an image downloading and caching library. When you use Picasso to download and display an image, the library keeps track of whether it has a copy of the image in memory or stored locally on the disk. The response time for retrieving and showing images in Picasso is very fast.

Picasso uses a context that is often your current `Activity`. In this example, the context is indicated by `this`, which refers to the `Activity`. Basic usage for displaying an image from a resource file into an `ImageView` is as follows:

```
Picasso.with(this).load(R.drawable.ic_launcher).into(imageView);
```

There are many other methods for Picasso; you can learn more at http://square.github.io/picasso/.

▼ TRY IT YOURSELF

Installing and Using Picasso

Picasso enhances your app when you work with images and bitmaps. These are the steps to download and use Picasso:

1. Go to http://square.github.io/picasso/ to learn more about Picasso.
2. In Android Studio, in Project view, find the app folder and locate the build.gradle file.
3. Add the line **compile** ‘**com.squareup.picasso:picasso:2.5.0**’ to the dependencies. That adds the Picasso library to the project.
4. Create an `Activity` with an `ImageView`.
5. Display an image from the drawable resources folder into the `ImageView` using Picasso.

Summary

In this hour, you looked at `ImageViews` and `Bitmaps`. You learned how `ScaleType` is used to change how images display in `ImageViews` and saw how rotation can be used in `ImageViews`. You learned about the `Matrix` class. You handled the display of an unrealistically large image by reading the dimensions of the bitmap before displaying it. You drew the word “Hello” on `Canvas` and displayed it in an `ImageView` to learn about the relationship between an `ImageView`, `Canvas`, and `Bitmap`. Picasso, an open source image library, was introduced.

Q&A

Q. If I am developing an app that displays images in a `ListView`, should I use `BitmapFactory.Options` to check the size of each image?

A. If you do not have control of the size of the images coming from the server, it is important to check size. If you do have control over the images, the ideal scenario is to have appropriately sized images. You can also use Picasso for handling images in code.

Workshop

Quiz

1. What is the purpose of `inJustDecodeBounds`?
2. What does mutable mean?
3. What is a pivot point?

Answers

1. In `BitmapFactory.Options`, `inJustDecodeBounds` is used to decode a `Bitmap` to get the dimensions but not actually create a `Bitmap` in memory.
2. Mutable means changeable, and that is important when you create `Bitmaps`. Some methods return mutable `Bitmaps` and others return immutable `Bitmaps`.
3. When you rotate an image, there must be a point to rotate around. That is the pivot point.

Exercise

For this exercise, use your own images or images that you find on the web. Create an `Activity` that includes two `ImageViews`. The first `ImageView` will take up the whole screen, with width and height set to `match_parent`. The second `ImageView` will have a fixed size. It will be smaller than the first `ImageView` and will appear over the first `ImageView` aligned on the bottom of the first `ImageView`. The goal is to use `scaleTypes` to display one image full size in the large view and to create a good thumbnail image in the smaller view. This exercise is an opportunity to experiment with displaying modified images. Try making the smaller image a set square size and use `CENTER_CROP` for the `scaleType`.

This page intentionally left blank

Index

A

accessing strings, 42

ActionBar class, applying, 207-213

actions, 32

- buttons, formatting, 97-102
- loaders, 326
- notifications, building with, 341

ACTION_VIEW, 195

activities, 5-6

- blank, 12, 67
- buttons, adding, 18
- code, adding to start, 21-23
- colors, formatting, 43
- configuring, modifying, 60-63
- data, passing data between, 24
- development, 7-8
- formatting, 20-21
- fragments, 53. *See also* fragments
  - formatting, 67-69
  - overview of, 66-70
- Intents class, starting, 17-27
- IntentService class, starting from, 81
- layouts, applying, 54
- lifecycles, 63-66
- with ListViews, 121
- MainActivity
  - checking for saved data, 62
  - formatting, 55
- MessageActivity, 55, 58-59
- overview of, 53-63
- results, returning, 54-60
- Settings, generating, 256
- starting, 23
- viewing, 18

Activity class, 8

adapters, 119

- ArrayAdapters, applying, 120-121
- BaseAdapters, 124-130
- view holder patterns, implementing, 131-135

AdapterViewFlipper class, applying, 172-174

adding

animation to FABs (floating action buttons), 158

apps on Google Play, 383

blank activities, 12

BroadcastReceiver to IntentService class, 81-86

buttons to activities, 18

CardView, 146-148

child views

BaseAdapters, 126-130

to toolbars, 217-219

code to start activities, 21-23

data

to databases, 266-267

for the Spinner, 162-163

destinations, 340

elevation, 145

fields to databases, 270

hints, 96

icons

to menu items, 211

to notifications, 341

images to ImageButton class, 98

imports on the fly, 22

MediaController to VideoViews, 197

navigating, 207

notifications

to chronometers, 343

Wear (Android), 353-354

Show Map buttons, 30

strings

resources, 100

viewing in TextView, 94

subscreens, 254

TextView components, 93

titles, 254

toolbars, 216

up navigation, 213

Add() method, 69

AddToBackStack() method, 69

ad supported applications, 384

aligning

to parents, 108-110

views, RelativeLayout, 110-112

Alpha levels, configuring, 185

alternative resources, 48

Amazon, publishing on, 384

Android

Calendar app and, 298-299

defining, 3-5

developer documentation, 369-373

locales, handling, 50

manifest files, 6, 13

platforms, 347-348

TV. *See* TV (Android)

updating, 369

Wear. *See* Wear (Android)

Android Asset Studio, formatting icons, 377

AndroidManifest.xml file, 80, 315, 376-377

Android Studio, 4, 10-14

activities, starting, 17-27

content providers, formatting, 312

installing, 11

IntentServices class, applying for, 86-87

layouts, formatting, 106-107

ListViews, 119

package files, exporting, 378

projects, formatting, 11-14

Settings activity, generating, 256

Show Map buttons, adding, 30

UI (user interface) controls, 91-92

user interfaces, formatting, 18-19

wearable apps, formatting, 349

Android Virtual Device. *See* AVD

animation, 143

to FABs (floating action buttons), adding, 158

sliding drawer, 220

APIs (application programming interfaces), 4, 234

Flickr APIs, 278

OpenGL ES graphics, 372

remote. *See* remote APIs

wearable apps, 353

Apple, iCal, 298

application programming interfaces. *See* APIs

applications (apps)

activities, 5

Calendar app, 298-299

CalendarContract content provider, 300-307

databases, applying, 269-274

emulators, running in, 23

evolution of app design, 139-140
localization, 50
private storage, 236-239
public storage, 240-241
publishing, 375
Amazon, 384
Google Play Services, 382
monetizing, 384-385
preparing for release, 375-382
sharing, 382-384
remote APIs, formatting, 286-290
resources
referencing, 40-41
viewing, 39
responsive, 73. *See also* background processes
signing, 378
themes, configuring, 145
wearable apps, 347

applying
ActionBar, 207-213
activities, 53-63
AdapterViewFlipper, 172-174
Android Studio
for IntentServices class, 86-87
UI (user interface) controls, 91-92
ArrayAdapters, 120-121
AsyncTask class, 76-78
BitmapFactory.Options class, 187-189
bitmaps, 186-191
Bluetooth, 373
colors, 42-43
Crashlytics, 369
databases, 269-274
dimensions, 44-45
drawable images, 45-46
EditText, 96
Fabric, 369
fragments, 70
Framelayout, 116
Google Play Services, 361-364
GridView, 171
implicit Intents, 27-32
languages, 49-51
layouts, 54
LinearLayout, 115
locations, 364-367
OnItemClickListener, 122-123
OpenGL ES graphics APIs, 372
padding, 113
Picasso, 192, 368
PieDbAdapter class, 270, 312
Pie objects, 262
queries, 267-269
Realm, 369
RecycleView, 148-154
resources, 40-48
SDKs (software development kits), 368-369
sensors, 370-371
services, background processes, 79-87
SharedPreferences, 245
SimpleCursorAdapter class, 271-274
StartActivityForResult, 54-60
strings, 41-42
styles, 47-48
toolbars, 214-219
<uses-feature> tag, 377
VideoViews, 196-197
View.post() method, 76

AppThemeNoActionBar style, 216

ArrayAdapters class, 170

arrays, adding data to resources, 123

AsyncTask class, 76-78, 282, 287

attributes
layouts, 112-113
showAsAction, 209

audio, playing MediaPlayer, 202-204

AutoCompleteTextView class, 164

AVD (Android Virtual Device), 347
TV (Android), 356
Wear (Android) apps, 351

B

backgrounds
downloading in the, 287
processes, 73-76
AsyncTask class, 76-78
post() method, 75-76
services, 79-87
UI (user interface) threads, 73-74

BaseAdapters class, 124-130

Beam, 373

bindView() method, 332

BitmapFactory.Options class, 187-189

bitmaps, 179, 186-191

blank activities, 12, 67

blocking UI threads, 74

Bluetooth, applying, 373

BroadcastReceivers class, 7

broadcasts, sending, 7

Bundle class, 62

buttons

- actions, formatting, 97-102
- activities, adding, 18
- clicking, 22, 99-102
- properties, 97-98

C

caches

- directories, 239
- storage, 235. *See also* file systems

Calendar app, 298-299

CalendarContract content provider, 300-307

calls, remote APIs, 278

cameras, 372

Canvas class, 186-191

cards, 144

CardView class, 146-148, 332-333

certificates, exporting files, 381

CheckBox class, 164-166

CheckBoxPreference class, 250-252

child views, 108

- BaseAdapters, adding, 126-130
- modifying, 123-124
- toolbars, adding, 217-219

chronometers, adding notifications, 343

classes, 8-10

- ActionBar
  - applying, 207-213
  - up navigation, 213
- Activity, 8
- AdapterViewFlipper, applying, 172-174
- ArrayAdapters, 120-121, 170
- AsyncTask, 76-78, 282, 287
- AutoCompleteTextView, 164
- BaseAdapters
  - adding child views, 126-130
  - formatting, 125-126
- BitmapFactory.Options, 187-189
- BroadcastReceiver, 10
  - adding IntentService class, 81-86
  - IntentService class, adding, 81-86
- BroadcastReceivers, 7
- Bundle, 62
- Canvas, 186-191
- CardView, 146-148, 332-333
- CheckBox, 164-166
- CheckBoxPreference, 250-252
- ContentProvider, 311-312
- CursorAdapter, 327-329, 330-332
- CursorLoader, 297, 324, 327-329
- CursorLoaders, 323
- DelayService, 81
- EditText, 96
- EditTextPreference, 253
- Framelayout, 114
- GestureDetector, 371
- GooglePlex, 31
- GridLayout, 105
- GridView, 171
- HttpUrlConnection, 280-281
- ImageButton, adding images, 98
- ImageViews, 179
  - navigating, 179-185
  - ScaleType property, 180
- Intents, 6, 9, 17-27
- IntentService, 9, 79-87
- IntentServices, 6
- LinearLayout, 105, 114-115
- ListPreference, 252-253
- ListViews, 119, 170
  - adding child views, 126-130
  - applying OnItemClickListener, 122-123
  - ArrayAdapters, 120-121
  - display information, 127
  - modifying child views, 123-124
  - overview of, 119-124
  - populating, 122
  - viewing data, 122

Loader, 324
LoaderManager, 324
LoaderManagerCallbacks, 325
loaders, 324-325
MainActivity, 17, 340
  checking for saved data, 62
  formatting, 55, 57
MediaController, adding VideoViews, 197
MediaPlayer
  objects, 202
  playing audio, 202-204
MessageActivity, 55, 58-59
MyContentProvider, 314, 319-321
Notification, 338
Notification.Builder, 338-339
NotificationManager, 338
notifications, 338
PendingIntent, 338
PhotoCursorAdapter, 331
PieDbAdapter, 263-266, 270, 312
PlaceholderFragment, 69
PreferenceActivity, generating, 256
PreferencesFragment, 249-250
ProgressBar, 78, 166-170
RadioButtons, 164-166
RecycleView, 146, 148-154
RelativeLayout, 93, 106, 114
  aligning views, 110-112
  navigating, 108-112
RemoteViews, 344
RippleDrawable, 157
ScrollView, 177
SecondActivity, 20-21
SecondaryActivity, 17
SeekBars, 166-170
Service, 79
SharedPreferences
  applying, 245
  configuring user preferences, 249-257
  data types, 248
  methods, 248
  reading from, 246-248
  storage, 245-248
SimpleCursorAdapter, 271-274, 300
Spinner, 161, 217
  adding data, 162-163
  retrieving data, 163
SQLiteOpenHelper, managing data, 263-266
SwitchPreference, 252-253
TableLayout, 114
TextView, 37, 41
  adding components, 93
  aligning to parents, 110
  applying styles, 48
  positioning components, 93
  UI (user interface) controls, 91
  viewing strings, 94
UriMatcher, 314
VideoViews, 195, 204
  adding MediaController, 197
  applying, 196-197
  classes, 204
  handling events, 199-202
  pausing video, 198
  playing audio with MediaPlayer, 202-204
  playing video, 195-199
  positioning video, 198
  starting video, 198
WearDestinationActivity, 353
WebView, 174-177

clauses, where, 267

clicking
  buttons, 22, 99-102
  responding to, 211

cloud, 277
  applications (apps), formatting, 286-290
  connecting, 290-294
  JSON (JavaScript Object Notation), 282-286

code
  activities, adding to start, 21-23
  applications (apps), viewing resources, 39
  loaders, 333
  MainActivity.java, 207
  OnClick property, 101
  sliding drawer activities, 226

colors
  applying, 42-43
  design, 142
  formatting, 43
  text, modifying, 95

<color> tag, 42

command-line tools, signing apps, 380

Commit() method, 69

common resources, 40-48

communication, NFC, 373

components

- design, 91
- of notifications, 337
- TextView
  - adding, 93
  - positioning, 93

conditions, resources, 37. *See also* resources

configuring

- activities, modifying, 60-63
- Alpha levels, 185
- elevation, 145
- Google Play Services, 361-363
- preferences, 245-246
- sliding drawer navigation, 222-224
- themes, 145
- user preferences, 249-257
- wearable apps, 349

connecting

- cloud, 290-294
- Google Play Services, 363-364

constructors, 150

ContentProvider class, 311-312

content providers, 297

- Calendar app, 298-299
- CalendarContract, 300-307
- declaring, 314-315
- formatting, 311
  - applying PieDbAdapter class, 312
  - defining URIs, 312-319
  - specifying URIs, 311
- getType() method, 317
- MyContentProvider class, 319-321
- overview of, 297
- query() method, 315

controls, 161

- AutoCompleteTextView, 164
- CheckBox, 164-166
- data display, 170-174
- information collection, 161-166
- progress indicators, 166-170
- RadioButtons, 164-166
- UIs (user interfaces), 91
  - applying Android Studio, 91-92
  - creating buttons (for actions), 97-102
  - handling input, 92-96
- ViewGroup, 372

Crashlyticsm, applying, 369

createBitmap() methods, 191

- creating. *See* formatting

CursorAdapter class

- CursorLoader class, 327-329
- formatting, 330-332

CursorLoader class, 297, 324

- CursorAdapter class, 327-329
- overview of, 324

CursorLoaders class, 323

cursors, applying queries, 267-269

customizing

- notifications, 342, 344-345
- styles, 372
- themes, 372
- views, 372

D

data

- adding/deleting/updating, 266-267
- remote APIs, fetching, 277-282
- SQLiteOpenHelper class, managing, 263-266

databases, 261

- applications (apps), applying, 269-274
- data, adding/deleting/updating, 266-267
- fields, adding, 270
- queries, applying cursors, 267-269
- tables, managing, 261-262

data types, SharedPreferences class, 248

declaring content providers, 314-315

default resources, specifying, 50

defining

- Android, 3-5
- IntentService class, 80
- queries, 268-269
- URIs (Uniform Resource Identifiers), 312-319

DelayService class, 81

delete() method, 312, 318

deleting data from databases, 266-267

- density-independent pixel. *See* dp

design

components, 91

flat, 140

layouts, 106-107

materials, 139

3D space, 141

animation, 143

cards, 144

evolution of app design, 139-140

graphics, 142

implementing, 145-159

overview of, 140-144

shadows, 141

typography, 143

RelativeLayout, 112

TV (Android), 355

wearable apps, 348

desktops, 139

destinations, adding, 340

developers

documentation, 369-373

signing up as, 278

development

activities, 5-8

BroadcastReceivers, 7

environments, 4. *See also* IDEs

intents, 6

IntentServices, 6

overview of, 5

services, 7-8

TV (Android), 355-357

Wear (Android) apps, 348-354

dimensions, applying, 44-45

directories

cache, 239

resources, qualifiers, 48

display information, ListViews, 127

displays, fragments, 70

documentation, developers, 369-373

doInBackground() method, 76, 287

dots per inch. *See* dpi

downloading

Android Studio, 11

in the background, 287

dp (density-independent pixel), 45

dpi (dots per inch), 45

dragging and dropping

images, 48

TextView components, 93

drawable images, 40, 45-46. *See also* images

drawing on Canvas, 190-191

E

editing, 91, 107

EditText class, applying, 96

EditTextPreference class, 253

elevation, configuring, 145

emulators

applications, running in, 23

maps, applying, 29

environments, development, 4. *See also* IDEs

events. *See also* Calendar app

inserting, 307

lists of attendees, 306

VideoViews, handling, 199-202

viewing, 305

evolution of app design, 139-140

execute() method, 77

explicit Intents, 27

exporting

files, certificates, 381

package files, 378

extending

BaseAdapter, 119

PreferencesFragment, 249

external SDKs (software development kits), 368-369

external storage, 233-234. *See also* file systems

files, saving, 241

F

Fabric, applying, 369

FABs (floating action buttons), 141, 155-159

features, 361-367

fetching data, remote APIs, 277-282

fields, adding to databases, 270

filename rules, 40

files

Android manifest, 6, 13

AndroidManifest.xml, 80, 315, 376-377

certificates, exporting, 381

external storage, saving, 241

images, moving, 48

layouts, 40, 91

package

exporting, 378

testing, 381

resources, formatting, 42

storage, 261

XML, RelativeLayout, 106

file systems, 233

overview of, 233-236

private storage, 236-239

public storage, 240-241

findViewById() method, 54, 353

flat design, 140

Flickr APIs, 278, 282

floating action buttons. *See* FABs

folders, navigating resources, 39

fonts, sizing, 44

formatting

activities, 20-21

BaseAdapters, 125-126

buttons (for actions), 97-102

colors, 43

content providers, 311

applying PieDbAdapter class, 312

defining URIs, 312-319

specifying URIs, 311

CursorAdapter class, 330-332

databases, applications (apps), 269-274

display information, ListViews, 127

fragments, activities, 67-69

icons, Android Asset Studio, 377

Intents with latitudes/ longitudes, 28-29

JSON (JavaScript Object Notation), 282-286

JSONArray, 283-284

JSONObject, 283

layouts, 106-107, 113

loaders, 326

MainActivity, 55, 57

MessageActivity, 58

notifications, 338-343

projects, Android Studio, 11-14

queries, 274

remote APIs, 286-290

resources, files, 42

sliding drawer navigation, 222-224

strings, 41

TV (Android), 355-357

URIs (Uniform Resource Identifiers), 297

user interfaces, 18-19

Wear (Android) apps, 348-354

forms, RelativeLayout, 112

fragments

activities, 53

formatting, 67-69

lifecycles, 63-66

overview of, 53-63

layouts, applying, 70

lists, viewing, 289

overview of, 66-70

FragmentTransction object, 69

Framelayout class, 114, 116

free applications, 384

functionality, 10

G

generating

PreferenceActivity class, 256

Settings activity, 256

signed APKs, 378

geo Intent, viewing GooglePlex, 31

GestureDetector class, 371

gestures, handling, 371

getActiveNetworkInfo() method, 290

getApplicationContext() method, 23

getAssets() method, 204

getCheckedRadioButtonID() method, 166

getColumnIndex() method, 269

getExternalFilesDir() method, 240-241

getItemCount() method, 150

getLoaderManager() method, 325

getPieFromCursor() method, 269

getResources() method, 40

getSharedPreferences() method, 246

getString() method, 42

getText() method, 61, 96, 164

getType() method, 312, 317

getView() method, 131

global positioning system. *See* GPS

Google
- Calendar, 298
- Play Services
  - applying, 361-364
  - configuring, 361-363
  - connecting, 363-364
  - locations, 364-367
  - publishing on, 382

GooglePlex class, 31

GPS (global positioning system), 364. *See also* locations

graphics, 142

gravity, 115

GridLayout class, 105, 114

GridView class, applying, 171

guidelines for images, 142

H

handling
- buttons, clicking, 99-102
- events, VideoViews, 199-202
- implicit Intents, 32-34
- large images, 186-187
- locales, 50
- selection, 224
- users
  - gestures, 371
  - input, 92-96

hasExtra() method, 33

Hello World, 37, 40

hints, adding, 96

holder patterns, implementing views, 131-135

horizontal scrolling, 174

HTTP (Hypertext Transfer Protocol), 278

HttpUrlConnection class, 280-281
- Hypertext Transfer Protocol. *See* HTTP

I

icons
- Android Asset Studio, formatting, 377
- menu items, adding, 211
- notifications, adding to, 341

IDEs (integrated development environments), 4. *See also* Android Studio

ImageButton class, 98

images
- drawables, 40
- guidelines, 142
- ImageButton, adding, 98
- large, handling, 186-187
- moving, 48
- rotating, 183-185
- sizing, 44
- viewing, 179-180

ImageViews class, 179
- navigating, 179-185
- ScaleType property, 180

implementing
- bindView() method, 332
- BroadcastReceiver, 82
- material design, 145-159
- newView() method, 330
- onCreateLoader() method, 329
- onLoaderReset() method, 329
- onLoadFinished() method, 329
- view holder patterns, 131-135

implicit Intents, 27
- applying, 27-32
- handling, 32-34

importing
- FABs (floating action buttons), 156
- on the fly, adding, 22

in-app payments, 384

<include/> element, 106

information collection controls, 161-166

Input controls, 92

inputType properties, 96

inserting events, 307

insert() method, 312, 318

installing
- Android Studio, 11
- Picasso, 192
- integrated development environments. *See* IDEs

Intents
- explicit, 27
- implicit, 27
  - applying, 27-32
  - handling, 32-34
- latitudes/longitudes, creating with, 28-29
- video, playing, 195

Intents class, 6, 9, 17-27

IntentService class, 9, 79-87
- activities, starting from, 81
- AndroidManifest.xml file, updating, 80

BroadcastReceiver, adding, 81-86

defining, 80

IntentServices class, 6, 86-87

interfaces, 5. *See also* APIs

internal padding, 113

internal storage, 233-234. *See also* file systems

internationalization, 49-51

items, adding icons, 211

J

Java. *See also* code

applications (apps), viewing resources, 39

MainActivity.java code, 207

Java Development Kits. *See* JDKs

JavaScript Object Notation. *See* JSON

JDKs (Java Development Kits), 380

JSON (JavaScript Object Notation), 282-286

JSONArray, formatting, 283-284

JSONObject, formatting, 283

K

keys

applications, signing, 378

stores, formatting, 381

L

languages, 49-51

large images, handling, 186-187

latitudes, creating with intents, 28-29

launching. *See* starting

layout_gravity property, 115

layouts, 105

activities, applying, 54

attributes, 112-113

editing, 107

files, 40, 91

formatting, 106-107

fragments, applying, 70

ListViews, display information, 127

margins, 113

notifications, 344

overview of, 105-108

RelativeLayout, navigating, 108-112

resources, programming, 108

sliding drawer navigation, 220-222

types, 114-116

layout_weight property, 115

lifecycles, activities, 63-66

LinearLayout class, 105, 114-115

listening for states, ViewViews, 199-202

ListPreference class, 252-253

lists

event of attendees, 306

fragments, viewing, 289

ListViews class, 119, 170

ArrayAdapters, applying, 120-121

child views

adding, 126-130

modifying, 123-124

data, viewing, 122

display information, 127

OnItemClickListener, applying, 122-123

overview of, 119-124

populating, 224

Loader class, 324

LoaderManagerCallbacks class, 325

LoaderManager class, 324

loaders, 323

actions, 326

classes, 324-325

code, 333

CursorAdapter class, formatting, 330-332

formatting, 326

overview of, 324

resetting, 326

states, 325-330

locales, handling, 50

locations

Google Play Services, 364-367

maps, opening, 28

tracking, 365-367

Lollipop, 4. *See also* OS (operating systems)

longitudes, creating with intents, 28-29

M

MainActivity class, 17, 340
- formatting, 55, 57
- saved data, checking for, 62

MainActivity.java code, 207

makePies() method, 127

managing
- data, SQLiteOpenHelper class, 263-266
- databases with tables, 261-262
- notifications, 338-343

maps
- opening, 28
- viewing, 29-31

margins, layouts, 113

material design, 139
- 3D space, 141
- animation, 143
- cards, 144
- evolution of app design, 139-140
- graphics, 142
- implementing, 145-159
- overview of, 140-144
- shadows, 141
- typography, 143

media, VideoViews, 195
- adding MediaController to, 197
- applying, 196-197
- audio with MediaPlayer, 202-204
- classes, 204
- handling events, 199-202
- pausing video, 198
- playing video, 195-199
- positioning video, 198
- starting video, 198

MediaController class
- VideoViews
  - adding, 197

MediaPlayer class
- audio, playing, 202-204
- objects, 202

memory, OOM (out-of-memory), 187

menus
- icons, adding, 211
- overflow, 209

MessageActivity class, 55, 58-59

methods
- Add(), 69
- AddToBackStack(), 69
- bindView(), 332
- Commit(), 69
- ContentProvider class, 312
- createBitmap(), 191
- delete(), 312, 318
- doInBackground(), 76, 287
- execute(), 77
- findViewById(), 54, 353
- geSharedPreferences(), 246
- getActiveNetworkInfo(), 290
- getApplicationContext(), 23
- getAssets(), 204
- getCheckedRadioButtonID(), 166
- getColumnIndex(), 269
- getExternalFilesDir(), 240-241
- getItemCount(), 150
- getLoaderManager(), 325
- getPieFromCursor(), 269
- getResources(), 40
- getString(), 42
- getText(), 61, 96, 164
- getType(), 312, 317
- getView(), 131
- hasExtra(), 33
- insert(), 312, 318
- makePies(), 127
- moveToFirst(), 268
- moveToLast(), 268
- moveToNext(), 268
- moveToPrevious(), 268
- newView(), 330
- onActivityCreated(), 325
- onBindViewHolder(), 150
- OnClickListener(), 21
- onClickListener(), 30, 99
- onConnected(), 367
- onConnectionFailed(), 363
- OnCreate(), 13, 33, 53
- onCreate(), 21, 64-66, 249, 312
- onCreateLoader(), 325, 329
- onCreateMenuOptions(), 211
- onCreateView(), 69
- onCreateViewHolder(), 150
- onHandleIntent(), 81
- onItemClickListener(), 122-123, 303
- onLoaderReset(), 329
- onLoadFinished(), 329
- onOptionsItemSelected(), 211
- onPause(), 64-66, 203
- onPostExecute(), 77
- onPreExecute(), 76

OnPreparedListener(), 199
onProgressUpdate(), 77, 168
onResume(), 64-66
onStart(), 64-66, 364
onStop(), 64-66
openFileOutput(), 239
pause(), 199
post(), 75-76
postExecute(), 77
publishProgress(), 76, 167
query(), 312
Remove(), 69
Replace(), 69
resolveActivity(), 30
run(), 75
SaveInstanceState(), 61-62
seekTo(), 199
sendBroadcast(), 10
setAlpha(), 185
setAnchorView(), 197
setAutoCancel(), 338
setContentIntent(), 341
setContentView(), 54
setImageBitmap(), 179
setImageDrawable(), 179
setImageResource(), 179
setImageUri(), 179
setLargeIcon(), 341
setOnClickListener(), 74
setTextViewText(), 344
SetTransition(), 69
SharedPreferences class, 248
start(), 199
startActivity(), 6, 195
startService(), 81
SystemClock.sleep(), 74
update(), 312, 318
View.post(), 76

mobile platforms, 5

modifying

activities, configuring, 60-63
child views, 123-124
queries, 321, 330
ScaleType property, 181
text
colors, 95
strings, 94

monetizing applications (apps), publishing, 384-385

moveToFirst() method, 268

moveToLast() method, 268

moveToNext() method, 268

moveToPrevious() method, 268

moving

images, 48
TextView components, 93

MP3 audio files, playing, 203

MyContentProvider class, 314, 319-321

N

navigating

ActionBar, applying, 207-213
adding, 207
ImageViews, 179-185
RelativeLayout, 108-112
resources, 37
common, 40-48
folders, 39
sliding drawer navigation, 219-228
toolbars, applying, 214-219
up navigation, 213
near-field communication. *See* NFC

newView() method, 330

NFC (near-field communication), 373

Notification.Builder class, 338-339

Notification class, 338

NotificationManager class, 338

notifications, 337

chronometers, adding, 343
customizing, 342, 344-345
formatting, 338-343
overview of, 337-338
Wear (Android), adding, 353-354

O

objects

cursors, retreiving, 269
FragmentTransction, 69
JSON (JavaScript Object Notation), 282-286
MediaPlayer, 202
Pie, applying, 262

onActivityCreated() method, 325

onBindViewHolder() method, 150

onClickListener() method, 21, 30, 99

onClick property, 99

onCompletionListener, 200

onConnected() method, 367

onConnectionFailed() method, 363

onCreateLoader() method, 325, 329

onCreateMenuOptions() method, 211

onCreate() method, 13, 21, 33, 53, 64-66, 249, 312

onCreateViewHolder() method, 150

onCreateView() method, 69

onDestroy() method, 64-66

onHandleIntent() method, 81

onItemClickListener() method, 122-123, 303

onLoaderReset() method, 329

onLoadFinished() method, 329

onOptionsItemSelected() method, 211

onPause() method, 64-66, 203

onPostExecute() method, 77

onPreExecute() method, 76

onPreparedListener() method, 199

onProgressUpdate() method, 77, 168

onResume() method, 64-66

onStart() method, 64-66, 364

onStop() method, 64-66

on the fly, adding imports, 22

OOM (out-of-memory), 187

openFileOutput() method, 239

OpenGL ES graphics APIs, 372

opening maps, 28

open source SDKs (software development kits), 368-369

operating systems. *See* OSs

Options menu, viewing, 207-209

OS (operating systems), 3, 139

out-of-memory. *See* OOM

overflow menus, 209

owner data, viewing, 332

P

package files

exporting, 378

testing, 381

padding, 113

paging, 174

paid models, applications, 384

palettes, Android Studio, 91. *See also* Android Studio

parents, aligning to, 108-110

parsing JSON (JavaScript Object Notation), 284-286

passing data between activities, 24

patterns, view holder, 131-135

pause() method, 199

pausing video, 198

PendingIntent class, 338

personal computers. *See* PCs

PhotoCursorAdapter class, 331

Picasso

applying, 368

overview of, 192

PieDbAdapter class, 263-266, 270, 312

Pie objects, applying, 262

pixels, sizing, 44

PlaceholderFragment class, 69

platforms

Android, 347-348

mobile, 5

playing. *See also* viewing

audio, MediaPlayer, 202-204

video, 195-199

populating ListViews, 224

positioning

TextView components, 93

video, 198

postExecute() method, 77

post() method, 75-76

PreferenceActivity class, generating, 256

preferences

configuring, 245-246

reading, 256

SharedPreferences, applying, 245

types, 251

user, configuring, 249-257

PreferencesFragment class, 249-250

primary colors, design, 142

private storage, 235. *See also* file systems

applications (apps), 236-239

processes, background, 73-76

AsyncTask class, 76-78

post() method, 75-76

services, 79-87

UI (user interface) threads, 73-74

programming

layouts, resources, 108

ScaleType property, 181

ProgressBar class, 78, 166-170

progress indicators, 166-170

projects

Android Studio, formatting, 11-14

resources, 37-39

structures, viewing, 38

properties

buttons, 97-98

inputType, 96

layout_gravity, 115

layout_weight, 115

OnClick, 99

padding, 113

ScaleType, 180-183

textColor, 95

public storage, 235, 240-241. *See also* file systems

publishing

applications (apps), 375

monetizing, 384-385

preparing for release, 375-382

sharing, 382-384

systems, 7

publishProgress() method, 76, 167

Q

qualifiers, directories, 48

queries

cursors, applying, 267-269

defining, 268-269

formatting, 274

modifying, 321, 330

query() method, 312, 315

R

RadioButtons class, 164-166

READ_EXTERNAL_STORAGE permission, 235

reading

app data, 237

preferences, 256

from SharedPreferences class, 246-248

Realm, applying, 369

RecycleView class, 146, 148-154

referencing resources, applications (apps), 40-41

RelativeLayout class, 93, 106, 114

navigating, 108-112

views, aligning, 110-112

releasing applications, 375. *See also* publishing

remote APIs, 277

applications (apps), formatting, 286-290

calls, 278

connecting, 290-294

data, fetching, 277-282

JSON (JavaScript Object Notation) data, 282-286

RemoteViews class, 344

Remove() method, 69

Replace() method, 69

requests, 6

resetting loaders, 326

resolveActivity() method, 30

resources, 37

alternative, 48

applications (apps)

referencing, 40-41

viewing, 39

arrays, adding data to, 123

colors, 42-43

common, 40-48

default, specifying, 50

dimensions, 44-45

directories, qualifiers, 48

drawable images, 45-46

filename rules, 40

files, formatting, 42

folders, navigating, 39

languages, 49-51

layouts, programming, 108

projects, 37-39

sizing, 44

storage, 40

strings, 41-42, 100

styles, 47-48

responding to clicks, 211

responsive applications (apps), 73. *See also* background processes

results, returning activities, 54-60

retreiving

data

from HttpUrlConnection, 280-281

from a Spinner, 163

objects from cursors, 269

RadioButtons, 166

returning results, activities, 54-60

right-to-left languages. *See* RTL languages

RippleDrawable class, 157

Roboto, 143

rotating images, 183-185

RTL (right-to-left) languages, 109

rules, filenames, 40

run() method, 75

running

apps in emulators, 23

on UI threads, 74

S

SaveInstanceState() method, 61-62

saving

app data, 239

files, external storage, 241

ScaleType property, 180-183

scrolling, horizontal, 174

ScrollView class, 177

SDKs (software development kits), 11, 48, 368-369

SecondActivity class, 20-21

SecondaryActivity class, 17

SeekBars class, 166-170

seekTo() method, 199

selection, handling, 224

sendBroadcast() method, 10

sending broadcasts, 7

sensors, applying, 370-371

Service class, 79. *See also* background processes

services

background processes, 79-87

development, 7-8

Google Play Services

applying, 361-364

locations, 364-367

starting, 6

setAlpha() method, 185

setAnchorView() method, 197

setAutoCancel() method, 338

setContentIntent() method, 341

setContentView() method, 54

setImageBitmap() method, 179

setImageDrawable() method, 179

setImageResource() method, 179

setImageUri() method, 179

setLargeIcon() method, 341

setOnClickListener() method, 74

setTextViewText() method, 344

Settings activity, generating, 256

SetTransition() method, 69

shadows, 141

SharedPreferences class

applying, 245

data types, 248

methods, 248

reading from, 246-248

storage, 245-248

user preferences, configuring, 249-257

sharing applications (apps), 382-384

showAsAction attribute, 209

showing. *See* viewing

Show Map buttons, adding, 30

signing

applications (apps), 378

up as developers, 278

SimpleCursorAdapter class, 271-274, 300

sizing. *See also* dimensions

resources, 44

text, 96

skeuomorphism, 139

sliding drawer navigation, 219-228

software development kits. *See* SDKs

specific locations, opening maps, 28

specifying

default resources, 50

language-specific resources, 51

URIs (Uniform Resource Identifiers), 311

Spinner class, 161-163, 217

SQL (Structured Query Language), 261

SQLite, 261

SQLiteOpenHelper class, 263-266

Square, 368

StartActivityForResult, 54-60

startActivity() method, 6, 195

starting

- activities, 23
  - adding to start, 21-23
  - Intents class, 17-27
- IntentService class from activities, 81
- services, 6
- video, 198

start() method, 199

startService() method, 81

statements, switch, 225

states

- FABs (floating action buttons), 155
- loaders, 325-330
- ViewViews, listening for, 199-202

storage. ***See also*** **file systems**

- files, 261
- resources, 37, 40
- SharedPreferences class, 245-248

stores, creating key, 381

strings, 37

- adding, viewing in TextView, 94
- applying, 41-42
- definitions, 40
- formatting, 41
- resources, adding, 100
- text, modifying, 94

<string> tag, 41

- Structured Query Language. *See* SQL

structures, viewing projects, 38

styles. ***See also*** **formatting**

- applying, 47-48
- AppThemeNoActionBar, 216
- customizing, 372
- text, 96

subscreens, adding, 254

subscribing systems, 7

support

- ad supported applications, 384
- color formats, 43

SwitchPreference class, 252-253

switch statements, 225

SystemClock.sleep() method, 74

T

TableLayout class, 114

tables, managing, 261-262

tags

- <color>, 42
- <string>, 41
- <uses-feature>, 377

testing

- package files, 381
- TV (Android), 357
- Wear (Android) apps, 351

text

- colors, modifying, 95
- sizing, 96
- strings, modifying, 94
- styles, 96

textColor property, 95

TextView class, 37, 41

- components
  - adding, 93
  - positioning, 93
- parents, aligning to, 110
- strings, viewing, 94
- styles, applying, 48
- UI (user interface) controls, 91

themes

- configuring, 145
- customizing, 372

threads, UI (user interface), 73-74

titles, adding, 254

toolbars

- adding, 216
- applying, 214-219
- child views, adding, 217-219

tools, updating, 24

tracking locations, 365-367

troubleshooting connecting, 290-294

TV (Android), 347

- AVD (Android Virtual Device), 356
- developing, 355-357
- as platforms, 347-348
- testing, 357

types

- layouts, 114-116
- preferences, 251
- of resources, 37

typography, 143

U

UIs (user interfaces), 5. *See also* interfaces
- controls, 91
  - applying Android Studio, 91-92
  - creating buttons (for actions), 97-102
- formatting, 18-19
- input, handling, 92-96
- reading/writing files, 236
- threads, 73-74
- updating, 19, 77
- Uniform Resource Identifiers. *See* URIs

uninstalling package files, 382

update() method, 312, 318

updating
- Android, 369
- AndroidManifest.xml file, 80, 315
- Calendar app, 307
- data in databases, 266-267
- tools, 24
- UI (user interface), 19, 77

up navigation, 213

UriMatcher class, 314

URIs (Uniform Resource Identifiers), 297
- defining, 312-319
- specifying, formatting content providers, 311

user gestures, handling, 371

user input, handling, 92-96

user preferences, configuring, 249-257

<uses-feature> tag, applying, 377

V

values, 37, 209

versions, OS (operating systems), 4

videos
- pausing, 198
- playing, 195-199
- positioning, 198
- starting, 198
- viewing, 195

VideoViews class, 195
- applying, 196-197
- audio, playing with MediaPlayer, 202-204
- classes, 204
- events, handling, 199-202
- MediaController, adding, 197
- video
  - pausing, 198
  - playing, 195-199
  - positioning, 198
  - starting, 198

ViewGroup controls, 372

viewing
- activities, 18
- Calendar apps, 300
- data, 122
- data display, 170-174
- events, 305
- images, 179-180
- lists in fragments, 289
- maps, 29-31
- Options menu, 207-209
- owner data, 332
- ProgressBar, 78
- projects, structures, 38
- resources, applications (apps), 39
- strings in TextView, 94
- videos, 195
- web pages, 31

View.post() method, applying, 76

views, 161, 174-177
- child, 108*. See also* child views
  - modifying, 123-124
- customizing, 372
- holder patterns, implementing, 131-135
- ListViews, overview of, 119-124
- RelativeLayout, aligning, 110-112
- ScrollView, 177
- WebView, 174-177

W

wearable apps, 347
- APIs (application programming interfaces), 353
- design, 348

Wear (Android) apps, 347-354

AVD (Android Virtual Device), 351

notifications, adding, 353-354

testing, 351

WearDestinationActivity class, 353

WebView class, 174-177

weight, 115

where clauses, 267

Widgets, 91

WRITE_EXTERNAL_STORAGE permission, 235

writing app data, 237

X

XML files

layouts, editing, 107

RelativeLayout, 106

This page intentionally left blank

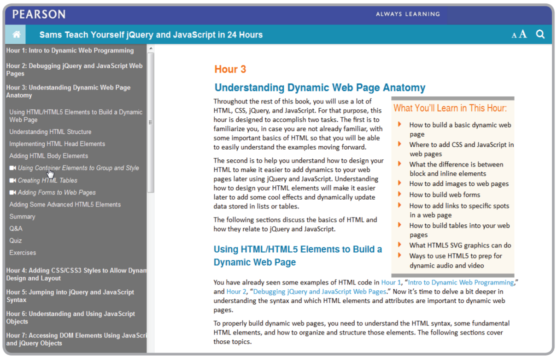
Learning Labs!

Learn online with videos, live code editing, and quizzes

SPECIAL 50% OFF – Introductory Offer
Discount Code: STYLL50

FOR A LIMITED TIME, we are offering readers of **Sams Teach Yourself** books a **50% OFF** discount to **ANY online Learning Lab** through Dec 15, 2015.

Visit **informit.com/learninglabs** to see available labs, try out full samples, and order today.


PEARSON
ALWAYS LEARNING
Sams Teach Yourself jQuery and JavaScript in 24 Hours
Hour 1: Intro to Dynamic Web Programming
Hour 2: Debugging jQuery and JavaScript Web Pages
Hour 3: Understanding Dynamic Web Page Anatomy
Using HTML/HTML5 Elements to Build a Dynamic Web Page
Understanding HTML Structure
Implementing HTML Head Elements
Adding HTML Body Elements
Using Container Elements to Group and Style
Creating HTML Tables
Adding Forms to Web Pages
Adding Some Advanced HTML5 Elements
Summary
Q&A
Quiz
Exercises
Hour 4: Adding CSS/CSS3 Styles to Allow Dynam Design and Layout
Hour 5: Jumping into jQuery and JavaScript Syntax
Hour 6: Understanding and Using JavaScript Objects
Hour 7: Accessing DOM Elements Using JavaScri and jQuery Objects

Hour 3

Understanding Dynamic Web Page Anatomy

Throughout the rest of this book, you will use a lot of HTML, CSS, jQuery, and JavaScript. For that purpose, this hour is designed to accomplish two tasks. The first is to familiarize you, in case you are not already familiar, with some important basics of HTML so that you will be able to easily understand the examples moving forward.

The second is to help you understand how to design your HTML to make it easier to add dynamics to your web pages later using jQuery and JavaScript. Understanding how to design your HTML elements will make it easier later to add some cool effects and dynamically update data stored in lists or tables.

The following sections discuss the basics of HTML and how they relate to jQuery and JavaScript.

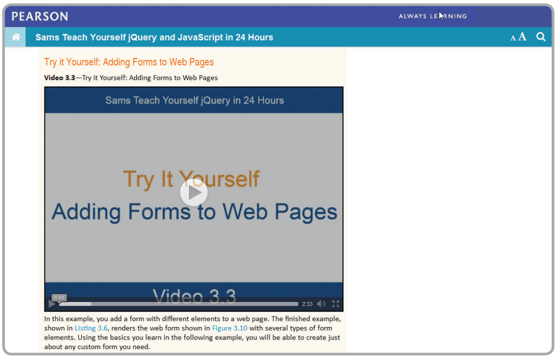
What You'll Learn in This Hour:

- How to build a basic dynamic web page
- Where to add CSS and JavaScript in web pages
- What the difference is between block and inline elements
- How to add images to web pages
- How to build web forms
- How to add links to specific spots in a web page
- How to build tables into your web pages
- What HTML5 SVG graphics can do
- Ways to use HTML5 to prep for dynamic audio and video

Using HTML/HTML5 Elements to Build a Dynamic Web Page

You have already seen some examples of HTML code in Hour 1, "Intro to Dynamic Web Programming," and Hour 2, "Debugging jQuery and JavaScript Web Pages." Now it's time to delve a bit deeper in understanding the syntax and which HTML elements and attributes are important to dynamic web pages.

To properly build dynamic web pages, you need to understand the HTML syntax, some fundamental HTML elements, and how to organize and structure those elements. The following sections cover those topics.


PEARSON
ALWAYS LEARNING
Sams Teach Yourself jQuery and JavaScript in 24 Hours
Sams Teach Yourself jQuery in 24 Hours
Try It Yourself
Adding Forms to Web Pages
Video 3.3
0:00
2:33

Try it Yourself: Adding Forms to Web Pages

Video 3.3—Try it Yourself: Adding Forms to Web Pages

In this example, you add a form with different elements to a web page. The finished example, shown in Listing 3.6, renders the web form shown in Figure 3.10 with several types of form elements. Using the basics you learn in the following example, you will be able to create just about any custom form you need.

- **Read** the complete text of the book online in your web browser
- **Watch** an expert instructor show you how to perform tasks in easy-to-follow videos

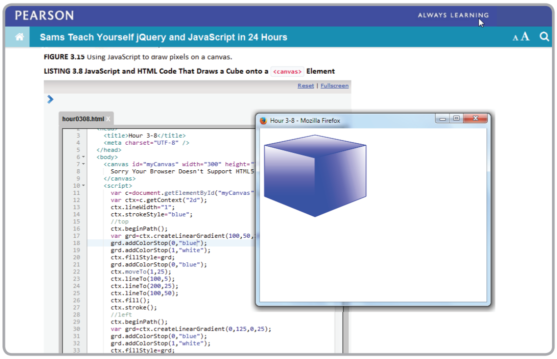
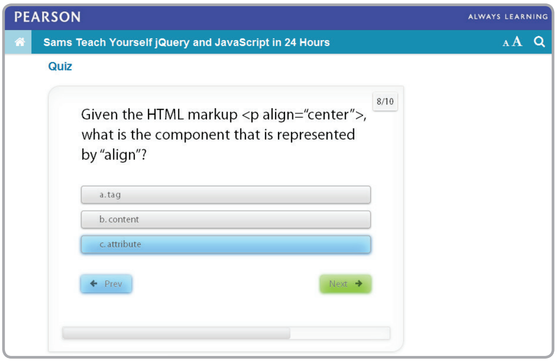

PEARSON
ALWAYS LEARNING
Sams Teach Yourself jQuery and JavaScript in 24 Hours
Reset | Fullscreen
hour0308.html
Hour 3-8 - Mozilla Firefox

FIGURE 3.15 Using JavaScript to draw pixels on a canvas.

LISTING 3.8 JavaScript and HTML Code That Draws a Cube onto a `<canvas>` Element

```
    <title>Hour 3-8</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="
      Sorry Your Browser Doesn't Support HTML5
    </canvas>
    <script>
      var c=document.getElementById("myCanvas"
      var ctx=c.getContext("2d");
      ctx.lineWidth="1";
      ctx.strokeStyle="blue";
      //top
      ctx.beginPath();
      var grd=ctx.createLinearGradient(100,50,
      grd.addColorStop(0,"blue");
      grd.addColorStop(1,"white");
      ctx.fillStyle=grd;
      grd.addColorStop(0,"blue");
      ctx.moveTo(1,25);
      ctx.lineTo(100,5);
      ctx.lineTo(200,25);
      ctx.lineTo(100,50);
      ctx.fill();
      ctx.stroke();
      //left
      ctx.beginPath();
      var grd=ctx.createLinearGradient(0,125,0,25);
      grd.addColorStop(0,"blue");
      grd.addColorStop(1,"white");
      ctx.fillStyle=grd;
```



Quiz

8/10

Given the HTML markup <p align="center">, what is the component that is represented by "align"?

a. tag
b. content
c. attribute

- **Try** your hand at coding in an interactive code-editing sandbox in select products
- **Test** yourself with interactive quizzes

ALWAYS LEARNING PEARSON

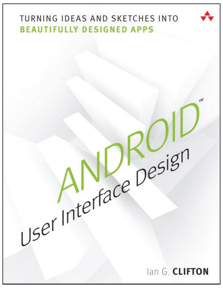
Essential Resources for Android Developers

informit.com/android

Developing Android User Interfaces LiveLessons (Video Training)

Adam Porter

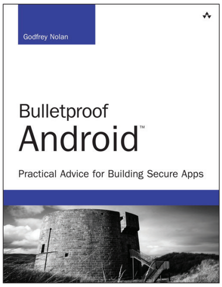
ISBN-13: 978-0-134-0-3773-8



Android User Interface Design

Ian G. Clifton

ISBN-13: 978-0-321-88673-6



Bulletproof Android: Practical Advice for Building Secure Apps

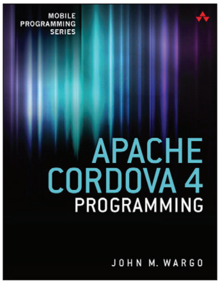
Godfrey Nolan

ISBN-13: 978-0-133-99332-5

Android Security Essentials LiveLessons (Video Training)

Godfrey Nolan

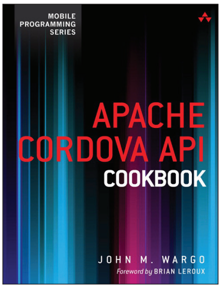
ISBN-13: 978-0-133-82904-4



Apache Cordova 4 Programming

John M. Wargo

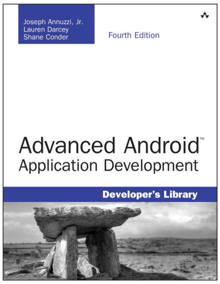
ISBN-13: 978-0-134-04819-2



Apache Cordova API Cookbook

John M. Wargo

ISBN-13: 978-0-321-99480-6



Advanced Android Application Development, 4th Edition

Joseph Annuzzi, Lauren Darcey, and Shane Conder

ISBN-13: 978-0-133-89238-3

Addison Wesley

PRENTICE HALL

Titles are available in print and/or eBook formats.

informIT® the trusted technology learning source

For more information and to read sample material, please visit informit.com/android.

Safari®

Titles are also available at safari.informit.com.