



B. M. Harwani

The Android Tablet Developer's Cookbook

Developer's Library



FREE SAMPLE CHAPTER

SHARE WITH OTHERS



The Android™ Tablet Developer's Cookbook

This page intentionally left blank

The Android™ Tablet Developer's Cookbook

B.M. Harwani

◆◆ Addison-Wesley

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

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.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data is on file.

Copyright © 2013 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. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

Google is a registered trademark of Google, Inc.

Android, Gmail, Google Currents, Google Maps, Google Play, and Nexus are trademarks of Google, Inc.

Amazon and Kindle Fire are registered trademarks of Amazon.com, Inc.

Java is a registered trademark of Oracle and/or its affiliates.

TOSHIBA is the trademark of Toshiba Corporation of Japan.

ASUS is a registered trademark of ASUSTeK Computer Inc.

Samsung, Galaxy, and Note are all trademarks of Samsung Electronics Co., Ltd.

Other names may be trademarks of their respective owners.

ISBN-13: 978-0-321-88530-2

ISBN-10: 0-321-88530-9

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

First printing: May 2013

Editor-in-Chief

Mark L. Taub

Executive Editor

Laura Lewin

Development

Editor

Songlin Qiu

Managing Editor

Kristy Hart

Project Editor

Betsy Gratrner

Copy Editor

Gill Editorial

Services

Indexer

Tim Wright

Proofreader

Debbie Williams

Technical

Reviewers

James Becwar

Romin Irani

Prashant Thakkar

Editorial Assistant

Olivia Basegio

Cover Designer

Chuti Prasertsith

Senior Compositor

Gloria Schurick



*Dedicated to my mother, Mrs. Nita Harwani,
and Dr. Martin Cooper.*

*Mom, you are a great teacher. When I look at you, I feel the
heights of love, patience, sacrifice, hard work, and emotions.*

*Dr. Martin Cooper, the inventor of the modern cell phone, has
revolutionized communication technology. His amazing and
dedicated research led to the development of cell phones that
interconnect the world even while moving. Today we can't
think of life without cell phones. Can we?*



This page intentionally left blank

Contents at a Glance

Introduction 1

I: User Interface Techniques

- 1 Overview of Android Tablet Applications 7
- 2 Fragments 47
- 3 ActionBars in Action 101
- 4 New Widgets 131

II: Managing Content

- 5 System Clipboard and Drag and Drop 155
- 6 Notifications and Pending Intents 183
- 7 Loaders 201

III: Multimedia Techniques

- 8 Animation 231
- 9 Hardware Accelerated 2D 281
- 10 Creating and Rendering Graphics 299
- 11 Recording Audio, Video, and Images 323

IV: Networking and Hardware Interface

- 12 Wireless Connectivity 361
- 13 Cores and Threads 391
- 14 Keyboards and Sensors 407

V: Exploring the Web

- 15 JSON 425
- 16 WebViews 449

VI: Advanced Android Techniques

| | | |
|----|-------------------------------------|-----|
| 17 | Adding Support for the Small Screen | 467 |
| 18 | Home Screen Widgets | 491 |
| 19 | Android Beam | 511 |
| 20 | Application Analytics and Tracking | 533 |
| | Index | 547 |

Acknowledgments

I am grateful to Laura Lewin, executive editor, for believing in me and giving me an opportunity to create this work.

My gratitude to Songlin Qiu who, as a development editor, offered significant feedback that shaped these chapters. She played a vital role in improving the structure and the quality of information.

I must thank James Becwar, Romin Irani, and Prashant Thakkar (Pandhi), the technical reviewers, for their excellent, detailed review of the work and the many helpful comments and suggestions they made.

Special thanks to Karen Gill, the copy editor, for first-class structural and language editing. I appreciate her efforts in enhancing the contents of the book and making it sound polished.

Big and ongoing thanks to Olivia Basegio, editorial assistant, for doing a great job, and sincere thanks to the whole team for getting the book published on time.

A great big thank-you to the editorial and production staff and the entire team at Pearson Technology Group who worked tirelessly to produce this book. I really enjoyed working with each of you.

I am also thankful to my family, who is my small world: Anushka, my wife, and my wonderful children, Chirag and Naman, for inspiring and motivating me and above all for forgiving me for spending so much time on my computer.

I must mention my dear students who have been a good teacher for me. Their interesting problems and queries have helped me understand reader expectations and inspired me to write books with a practical approach.

About the Author

B.M. Harwani is founder and owner of Microchip Computer Education (MCE), based in Ajmer, India, which provides computer education in all programming and Web developing platforms. He graduated with a BE in computer engineering from the University of Pune, and has a C Level (Master's Diploma in Computer Technology) from DOEACC, Government of India. Being involved in the teaching field for more than 19 years, he has developed the art of explaining even the most complicated topics in a straightforward and easily understandable fashion. He wrote *Android Programming Unleashed* (Sams Publishing), among other books, and has taught programming courses for 17 years. To know more, visit Harwani's blog at <http://bmharwani.com/blog>.

Introduction

Android is Google's open source and free Java-based platform for mobile development. Tablets are getting more popular every day. They are gadgets that fall between smartphones and personal computers. They have faster processors than smartphones but much less processing power than computers. They are handy and lightweight. From a tablet, you can watch videos, listen to music, surf the Web, make calls over Wi-Fi networks, read electronic documents, play games, and launch apps.

An Android tablet is a touch-screen, mobile device that runs the Android operating system. Google fueled the development of Android tablets with the release of Android 3.0 SDK. Actually, Android 3.0 SDK was designed with tablets in mind (that is, larger screen devices). The launch of Android 3.0 SDK compelled manufacturers around the world to produce Android tablets. As the market becomes flooded with Android tablets, developers are attracted to developing apps for the growing Android tablet market. Looking at the huge demand of developing applications for Android tablet inspired me to write this book.

Who Should Read This Book

Like any good book, this book provides easy-to-reuse code designed to solve real-world problems for intermediate to advanced users. The book covers the range from basic to complex problems that developers usually come across. The book will be beneficial for developers and instructors who want to learn or teach Android tablet programming. In short, it is a useful reference for anyone who wants to understand all key aspects of Android tablet programming and wants to find quick answers to different critical problems that appear while developing Android applications.

Key Topics That This Book Covers

This book is comprehensive and covers each topic in deep detail. Key topics covered include the following:

- Activities and Android activity life cycle
- Starting activity using intent and passing data from one activity to another
- Displaying and using `ListFragment`, `DialogFragment`, and `PreferenceFragment`
- Creating tabbed and drop-down list `ActionBar`
- Creating custom content providers
- Using frame-by-frame and tweening animation
- Using accelerometer, proximity, and gyroscope sensors
- Using JSON for managing data exchange formats
- Enabling and disabling Wi-Fi and using Wi-Fi Direct
- Using `RemoteViews` and creating and updating home screen widgets
- Creating and rendering graphics using OpenGL
- Cutting, copying, and pasting text using the system clipboard
- Reading and writing into NFC tags
- Transferring data using Android Beam
- Applying smooth coloring, rotating, scaling, and translating graphics
- Capturing image, audio, and video using built-in intent as well as Java code
- Using EasyTracker Library and GoogleAnalytics Singleton for tracking Android applications

The book is completely up to date with the latest Jelly Bean 4.2.

Key Benefits That This Book Provides

By the time you finish this book, you will be well versed with the following concepts:

- Adding and removing fragments with device orientation
- Implementing communication between fragments
- Displaying action items, action views, and the submenu in the `ActionBar`
- Creating a stack of images using `StackView`
- Displaying a list of options using `ListPopupWindow` and `PopupMenu`
- Dragging and dropping text and images
- Creating and using notifications
- Creating and using pending intent to start an activity
- Using loaders for accessing information

- Using `ValueAnimator` and `ObjectAnimator` to animate views
- Implementing multiple animations using `AnimatorSet`
- Applying layout animation
- Applying hardware acceleration for improving graphic system performance
- Using view layers and `SurfaceView` to improve graphics application performance
- Applying transformations using `TextureView`
- Transferring files through Bluetooth
- Using threads and multiple threads
- Using `WebView`, `WebViewClient` class, and `WebViewFragment` for displaying Web pages
- Supporting different platform versions
- Supporting older Android versions
- Adapting to screen orientations
- How Android tablet apps are different from Phone apps
- Porting apps between small-screen phones and tablets

How This Book Is Organized

This book is structured in six parts:

- **Part I: User Interface Techniques**

In Chapter 1, “Overview of Android Tablet Applications,” you will learn about the sizes and features of various Android tablets. You will also learn to create Android virtual devices and discover the directory structure of Android projects. In addition, you will learn about activities and different phases in the Android activity life cycle. You will learn how an activity is started using intent and how data is passed from one activity to another.

Chapter 2, “Fragments,” focuses on understanding the concept of fragments. You will learn the life cycle of a fragment and the procedure to create foreground fragments. You will also come to know the difference between foreground and background fragments and discover how to add and remove fragments with device orientation. You will learn how `FragmentManager` and `FragmentTransaction` handle fragments. In addition, you will learn to create fragments dynamically at runtime and implement communication between fragments. Finally, you will study the procedure to display options through `ListFragment`, display dialogs through `DialogFragment`, and set user preferences using `PreferenceFragment`.

Chapter 3, “ActionBars in Action,” focuses on understanding the usage of ActionBars in Android applications. You will learn the difference between the menu and the ActionBar. You will discover how to toggle the ActionBar’s visibility, understand its different

components, and walk through the procedure to display action items in the ActionBar. You will learn to display action views and submenus in an ActionBar. Finally, you will learn to create a tabbed and a drop-down list in an ActionBar.

Chapter 4, “New Widgets,” focuses on the working of new widgets. You will learn to display the Calendar in the Android application, display and select numbers through `NumberPicker`, and create a stack of images using `StackView`. You will also learn to display a list of options using `ListPopupWindow` and `PopupMenu`.

■ Part II: Managing Content

In Chapter 5, “System Clipboard and Drag and Drop,” you will learn how to drag and drop text and images. Also, you will learn to cut, copy, and paste text using the system clipboard.

In Chapter 6, “Notifications and Pending Intents,” you will learn about pending intents and how intents are broadcasted. You will also come to understand the Android notification system. You will learn to create notifications, use `Notification.Builder`, and obtain a `NotificationManager`. Finally, you will create notifications and use pending intent to start an activity.

Chapter 7, “Loaders,” focuses on understanding the concept of loaders. You will also learn about content providers and how to use `CursorLoader` to access information from a contact’s content provider. Finally, you will learn to create your own custom content providers.

■ Part III: Multimedia Techniques

In Chapter 8, “Animation,” you will learn about different types of animations. You will learn to use `ValueAnimator` and `ObjectAnimator` in animating views. You will learn to implement multiple animations using `AnimatorSet`. Also, you will study frame-by-frame animation and tweening animation. You will learn to apply layout animation and the procedure to collect and sequence animations using `AnimationSet`.

In Chapter 9, “Hardware Accelerated 2D,” you will learn to use hardware acceleration. You will learn to use view layers and improve performance of graphics-based applications using `SurfaceView`. Finally, you will learn to apply transformations using `TextureView`.

Chapter 10, “Creating and Rendering Graphics,” explains different APIs that are required for displaying graphics. You will learn to create and render a rectangle using OpenGL. Also, you will come to understand the difference between coloring a vertex and lighting. In addition to this, you will learn to apply smooth coloring and rotate, scale, and translate graphics.

Chapter 11, “Recording Audio, Video, and Images,” explains the technique of capturing an image using built-in intent and Java code. You will learn to record audio and video using built-in intent and through Java code, understand `CamcorderProfile`, `MediaRecorder`, and their methods.

■ Part IV: Networking and Hardware Interface

Chapter 12, “Wireless Connectivity,” focuses on how connections between devices are established wirelessly. You will learn to pair two Bluetooth-enabled devices, manually transfer files from one device to another using Bluetooth and the procedure, and pair a Bluetooth device with a Windows PC. You will also learn to enable a local Bluetooth device, display the list of paired devices, and transfer files through Bluetooth. Finally, you will learn the concept of Wi-Fi, discover how to enable and disable Wi-Fi, and understand the usage of Wi-Fi Direct.

In Chapter 13, “Cores and Threads,” you will learn the utility of multicore processor architectures. You will understand the utility of Garbage Collection (GC). You will also learn about threads and multiple threads. Finally, you will learn about the `AsyncTask` class.

In Chapter 14, “Keyboards and Sensors,” you will learn how to change Android keyboards and input methods. You will learn about sensors and display the list of sensors supported by a device. Finally, you will learn to use Accelerometer, Proximity, and Gyroscope sensors.

■ Part V: Exploring the Web

In Chapter 15, “JSON,” you will learn the concept of JSON. You will learn to use `JSONObject`, `JSONArray`, `JSONArrays`, and use `JSONArray` to keep information. Also, you will learn to use `JsonReader` and `JsonWriter`.

In Chapter 16, “WebViews,” you will learn to display Web pages using `WebView`. Also, you will learn to use the `WebViewClient` class and the `WebViewFragment`.

■ Part VI: Advanced Android Techniques

In Chapter 17, “Adding Support for the Small Screen,” you will learn about factors for supporting various screens and densities. You will learn how different platform versions are supported in Android applications and how Android Support Library is used to support older Android versions. You will learn to adapt to screen orientation by anchoring controls. Also, you will learn to define alternate layouts to handle screen orientation.

In Chapter 18, “Home Screen Widgets,” you will learn about `RemoteViews`, app widgets, and home screen widgets. You will come to understand the app widget life cycle methods. In addition, you will learn to create a home screen widget and update it through a `Button` control. Finally, you will learn to update a home screen widget using `AlarmManager`.

In Chapter 19, “Android Beam,” you will learn about Near Field Communication (NFC) and the role of NFC tags. Also, you will learn about the structures used in exchanging information with NFC Tags, read and write into NFC tags. Finally, you will come to understand Android beam and how data is transferred using it.

In Chapter 20, “Application Analytics and Tracking,” you will learn the concept of application analytics and tracking. You will learn to use `EasyTracker` Library and `GoogleAnalytics` Singleton to track Android applications.

Code Examples for This Book

All the Android recipes discussed in this book are available to download from the publisher's Web site at www.informit.com/title/9780321885302. Download the code bundle provided at the site and unzip it. Follow these steps to use the provided code:

1. Launch Eclipse.
2. Select File, Import. From the Import dialog that opens, select Existing Projects into Workspace and then click Next.
3. In the following dialog, click the Browse button to locate and select the folder where you unzipped the code bundle.
4. After you select the code bundle, all the Android projects enclosed in it will appear in the Projects box. By default, all the projects will be found checked. You can uncheck project(s) that you don't want to import and then click Finish. That's it. The projects will be imported into Eclipse and are ready to run.

Assumptions

The following three things are assumed in all the recipes developed in this book:

- Until specified, the `android:minSdkVersion` and `android:targetSdkVersion` attributes of all the apps in this book are assumed to be 11 and 17, respectively. API Level 11 and 17 refer to the Android 3.0 (Honeycomb) and Android 4.2 (Jelly Bean) respectively. It also means that the applications developed in this book require a minimum of API Level 11 to run. Also, the applications are compiled and designed to run on API Level 17.
- An XML file, `dimens.xml`, is assumed to be created in the `res/values` folder with the code as shown here:


```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="text_size">14sp</dimen>
</resources>
```
- Two folders, `values-sw600dp` and `values-sw720dp`, are assumed to be created in the `res` folder. Also, the `dimens.xml` file from the `res/values` folder is assumed to be copied into these two folders. To match the 7-inch and 10-inch tablet screens, the dimension resource `text_size` in the `dimens.xml` file in the `values-sw600dp` and `values-sw720dp` folders is assumed to be modified to `24sp` and `32sp`, respectively. For more information, refer to the recipe "Converting an Android Phone Application into an Android Tablet Application" in Chapter 1.

New Widgets

In this chapter, you are going to discover the new widgets that have become available since API 11 level. You will learn to display the calendar in the Android application through `CalendarView` and display a range of numbers through `NumberPicker`. You will also learn to display a stack of images using the `StackView` widget. Finally, you will learn to display a list of options using `ListPopupWindow` and display suggestions through `PopupMenu`.

Recipe: Displaying the Calendar in an Android Application

To display the calendar in an Android application, you will use `CalendarView`. This is a configurable widget that displays and selects dates. By default, the calendar of the current month is displayed, but you can scroll through to the desired date. To select a date, just tap on it.

To view a calendar, create an Android project called `CalendarViewApp`. The application will display a calendar of the current month by default. The user can scroll through the calendar to display dates of a particular month. After selecting a date, it will be displayed through `Toast`. The application will also contain a `Button` control that, when clicked, will display `DatePickerDialog`, allowing the user to display the calendar of the desired month.

Because your application needs a button and a calendar, you must define both `Button` and `CalendarView` in the activity layout file. After you define the `Button` and `CalendarView`, the activity layout file `activity_calendar_view_app.xml` will appear as shown in Listing 4.1.

Listing 4.1 Code Written in the Activity Layout File `activity_calendar_view_app.xml`

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button android:id="@+id/date_picker_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Open Date Picker"
        android:textSize="@dimen/text_size" />
    <CalendarView
        android:id="@+id/calendar_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>

```

To access and identify Java code, the `Button` and `CalendarView` are assigned the ID, `date_picker_button` and `calendar_view`, respectively. Next, you need to write Java code to perform the following tasks:

- Display the `CalendarView` defined in the activity layout file.
- Associate an event listener, `setOnClickListener`, to the `Button` control to display `DatePickerDialog`.
- Associate `OnDateSetListener` to `DatePickerDialog` to display the calendar of the selected date through `CalendarView`.
- Associate an event listener to the `CalendarView` to display the selected date on the screen.

To accomplish the preceding tasks, you write code as shown in Listing 4.2 in the Java activity file `CalendarViewAppActivity.java`.

Listing 4.2 Code Written in the Java Activity File `CalendarViewAppActivity.java`

```

package com.androidtablet.calendarviewapp;

import android.os.Bundle;
import android.app.Activity;
import android.widget.CalendarView;
import android.widget.CalendarView.OnDateChangeListener;
import android.widget.Toast;
import java.util.Calendar;

```

```

import android.app.DatePickerDialog;
import android.widget.DatePicker;
import android.widget.Button;
import android.view.View;
import android.view.View.OnClickListener;

public class CalendarViewAppActivity extends Activity {
    private CalendarView calendarView;
    private int yr, mon, dy;
    private Calendar selectedDate;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_calendar_view_app);
        Calendar c = Calendar.getInstance();
        yr = c.get(Calendar.YEAR);
        mon = c.get(Calendar.MONTH);
        dy = c.get(Calendar.DAY_OF_MONTH);
        Button datePickerButton = (Button) findViewById(
            R.id.date_picker_button);
        calendarView = (CalendarView) findViewById(
            R.id.calendar_view);
        datePickerButton.setOnClickListener(new
            OnClickListener() {
                public void onClick(View v) {
                    new DatePickerDialog(CalendarViewAppActivity.
                        this, dateListener, yr, mon, dy).show();
                }
            });
        calendarView.setOnDateChangeListener(new
            OnDateChangeListener() {
                @Override
                public void onSelectedDayChange(CalendarView view,
                    int year, int month, int dayOfMonth) {
                    Toast.makeText(getApplicationContext(), "Selected
                        date is "+(month+1)+"-"+dayOfMonth+"-"+
                        year, Toast.LENGTH_SHORT). show();
                }
            });
    }

    private DatePickerDialog.OnDateSetListener dateListener =
        new DatePickerDialog.OnDateSetListener() {

```

```

    public void onDateSet(DatePicker view, int year, int
        monthOfYear, int dayOfMonth){
        selectedDate=Calendar.getInstance();
        yr=year;
        mon=monthOfYear;
        dy=dayOfMonth;
        selectedDate.set(yr, mon, dy);
        calendarView.setDate(selectedDate.getTimeInMillis());
    }
};
}

```

You can see in the preceding code that the `CalendarView` with ID `calendar_view` is accessed from the layout file and is mapped to the `CalendarView` object `calendarView`. Also, the `Button` control with ID `date_picker_button` is accessed from the layout file and is mapped to the `Button` object `datePickerButton`. `setOnClickListener` is associated to the `Button` control, and its callback method, `onClick`, executes when the `Button` is clicked. In the `onClick` callback method, `DatePickerDialog` is invoked to display the current date.

The `onDateSetListener` is associated to the Date Picker dialog so that when any date is selected from the Date Picker dialog, the `CalendarView` is set to display the calendar of the selected month and year.

The `setOnDateChangeListener` is associated to the `CalendarView`. When any date is selected or changed in the `CalendarView`, the callback method `onSelectedDayChange()` is called. Using the `onSelectedDayChange()` method, you display the selected date through `Toast`. The thing to remember here is that the month is 0 based, so you must add 1 to it before displaying.

After running the application, you see the `CalendarView` displaying the calendar of the current month (see Figure 4.1 [top]). To see the calendar of the desired month, select the Open Date Picker button, which opens `DatePickerDialog`. From `DatePickerDialog`, you can select the date from the calendar (see Figure 4.1 [middle]). After selecting a date from `DatePickerDialog` and selecting Done, the calendar of the selected date will be displayed. Also, after you select a date from the `CalendarView`, it is displayed through `Toast`, as shown in Figure 4.1 (bottom).

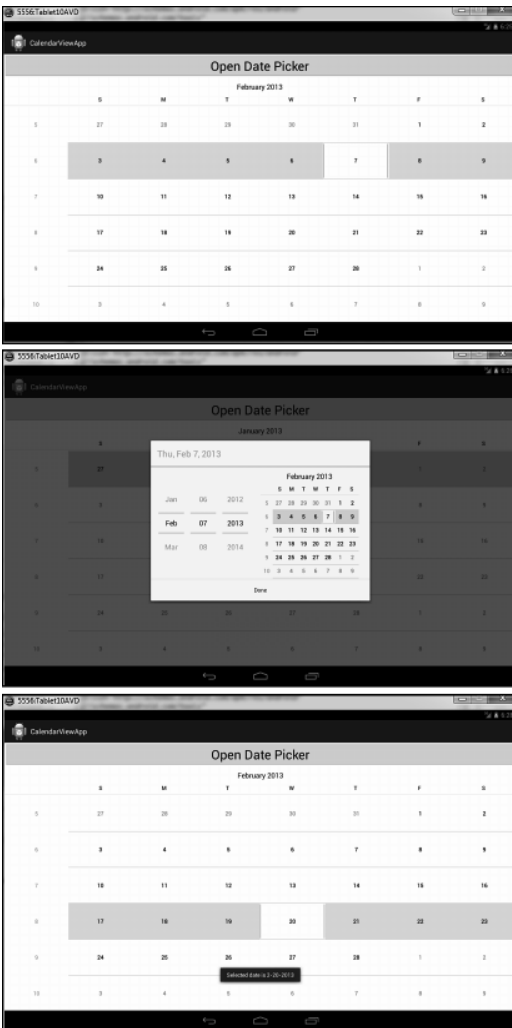


Figure 4.1 `CalendarView` showing the calendar of the current month (top); `DatePickerDialog` opens after selecting the `Open Date Picker` button (middle); `CalendarView` displays the calendar of the date selected from `DatePicker` (bottom).

Recipe: Displaying and Selecting Numbers Through `NumberPicker`

In this recipe, you will learn to display a `NumberPicker` that displays numbers in the specified range. The number that is selected from the `NumberPicker` is displayed through `TextView`. Create a new Android project called `NumberPickerApp`.

You just want to display `TextView` and `NumberPicker` in this application. The `NumberPicker` will display the numbers between the specified range, and the `TextView` will display the number selected from the `NumberPicker`. To define the `TextView` and `NumberPicker`, write the code shown in Listing 4.3 to the activity layout file `activity_number_picker_app.xml`.

Listing 4.3 Code Written in the Activity Layout File `activity_number_picker_app.xml`

```
<LinearLayout
    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:orientation="vertical" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Select a number from NumberPicker"
        android:id="@+id/numberview"
        android:textSize="@dimen/text_size"
        android:textStyle="bold" />
    <NumberPicker android:id="@+id/numberpicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

You can see that the `TextView` control is assigned the ID `numberview` and is initialized to display the text `Select a number from NumberPicker`. The text displayed through `TextView` will appear in bold and in the size defined by the dimension resource `text_size`. To access and identify Java code, assign `NumberPicker` the ID `numberpicker`.

In the main Java activity file, you need to write the Java code to do the following tasks:

- Access the `TextView` and `NumberPicker` from the layout file, and map them to the respective objects.
- Set the maximum and minimum numerical values to be displayed through `NumberPicker`.
- Associate an event listener to the `NumberPicker` to listen if the current value in `NumberPicker` is changed.
- Display the number selected from the `NumberPicker` through `TextView`.

To perform all these tasks, write the code shown in Listing 4.4 to the Java activity file `NumberPickerAppActivity.java`.

Listing 4.4 Code Written in the Java Activity File `NumberPickerAppActivity.java`

```

package com.androidtablet.numberpickerapp;

import android.os.Bundle;
import android.app.Activity;
import android.widget.NumberPicker;
import android.widget.TextView;

public class NumberPickerAppActivity extends Activity {
    TextView numberView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_number_picker_app);
        numberView = (TextView)findViewById(R.id.numberview);
        NumberPicker numberPicker = (NumberPicker) findViewById(R.id.numberpicker);
        numberPicker.setMaxValue(100);           #1
        numberPicker.setMinValue(0);             #2
        numberPicker.setWrapSelectorWheel(true);
        numberPicker.setOnValueChangedListener( new NumberPicker.
            OnValueChangedListener() {
                @Override
                public void onValueChange(NumberPicker picker, int
                    oldVal, int newVal) {
                    numberView.setText("Selected number is "+
                        newVal);
                }
            });
    }
}

```

You can see that the `TextView` control with ID `numberview` is accessed from the layout file and mapped to the `TextView` object `numberView`. Similarly, `NumberPicker` with ID `numberpicker` is accessed from the layout file and is mapped to the `NumberPicker` object `numberPicker`. The minimum and maximum values to be displayed through `NumberPicker` are set to 0 and 100, respectively.

The `setWrapSelectorWheel()` method is set to `true` to make the selector wheel wrap around the minimum and maximum values that are displayed through `NumberPicker`. When the range of values (that is, maximum value – minimum values) displayed through `NumberPicker` is more than the number of numerical shown in the selector wheel, wrapping is enabled by default. (The selector wheel wraps around the maximum and minimum values by default.)

The `setOnValueChangedListener` is associated with the `NumberPicker`. When the current value is changed in the `NumberPicker`, the callback method `onValueChange` is invoked. In the `onValueChange` method, the newly selected number in the `NumberPicker` is displayed through the `TextView` control.

After you run the application, the `TextView` will display a text message directing the user to `Select a number from NumberPicker`. The `NumberPicker` displays the assigned minimum value in editable form. The lesser value is shown above, and a greater value is shown below (see Figure 4.2 [top]). You can change the number by scrolling up or down and by tapping on the lesser or greater value shown above and below. When you tap a number, it is displayed through the `TextView` control, as shown in Figure 4.2 (bottom).

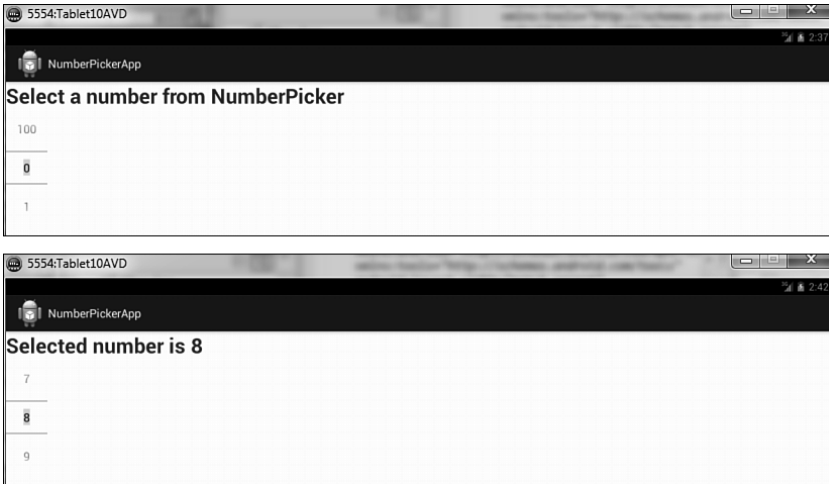


Figure 4.2 `NumberPicker` displaying the numbers beginning from the set minimum value (top) and the selected number displayed through `TextView` (bottom)

You can display any range of values through the `NumberPicker`. For example, to display odd values from 1 to 19, you can replace the statements #1 and #2 in Listing 4.4 with the following code:

```
String[] stringArray = new String[10];
int n=1;
for(int i=0; i<10; i++){
    stringArray[i] = Integer.toString(n);
    n+=2;
}
numberPicker.setMaxValue(stringArray.length-1);
numberPicker.setMinValue(0);
numberPicker.setDisplayedValues(stringArray);
```

You can see that a `String` array called `stringArray` is defined and values 1,3,5... 19 are stored in it. The min value of the `NumberPicker` is set to 0. The max value of `NumberPicker` is set equal to the length of `stringArray` -1 because you want to display all the elements of the array `stringArray`. Thereafter, through the `setDisplayedValues()` method, the values in the `stringArray` are displayed through `NumberPicker`.

Because the current theme in the Android application is derived from `Theme_Holo` or `Theme_Holo_Light`, the `NumberPicker` appears as shown in Figure 4.2 (that is, the current value as editable with lesser and greater value shown above and below the `NumberPicker`, respectively). If you change the theme of your application, you can change the appearance of the `NumberPicker`. For example, the following statements applied in the `AndroidManifest.xml` file will set the current theme of the application to be derived from `Theme`:

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Black.NoTitleBar" >
```

The preceding statements will change the theme of the application to `Theme.Black.NoTitleBar` and hence the appearance of the `NumberPicker` widget. In other words, the `NumberPicker` will display the current value in editable form with an increment and decrement button displayed above and below, respectively (see Figure 4.3 [top]). After you change the current value, it will be displayed through `TextView`, as shown in Figure 4.3 (bottom).

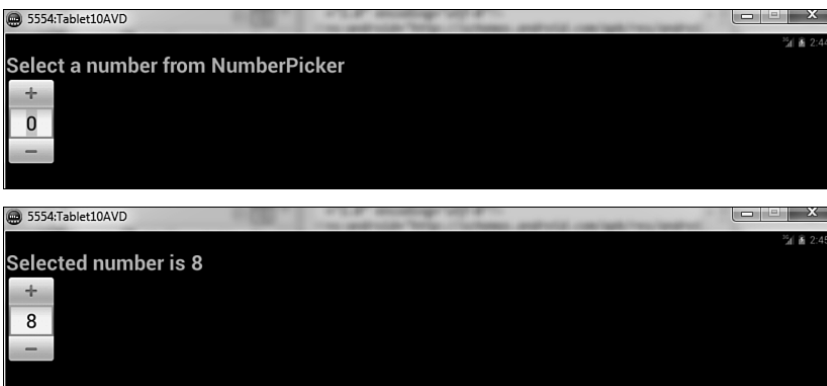


Figure 4.3 `NumberPicker` with black background, increment, and decrement buttons on changing the theme of the application (top), and the selected number displayed through `TextView` (bottom)

Recipe: Creating a Stack of Images Using StackView

`StackView` helps in arranging items in the form of stacked cards, where the front item can be flipped to bring the item behind it to the front. In addition to images, you can stack objects composed of text and other data, too.

In this recipe, you will learn to stack images in the `StackView`. So create a new Android project called `StackViewApp`. The only control that you need to define in the activity layout file is `StackView` widget. After defining the `StackView` widget, the activity layout file `activity_stack_view_app.xml` will appear, as shown in Listing 4.5.

Listing 4.5 Code Written in the Activity Layout File `activity_stack_view_app.xml`

```

<FrameLayout
    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" >
    <StackView
        android:id="@+id/stackview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:animateLayoutChanges="true">
    </StackView>
</FrameLayout>

```

To access and identify the `StackView` in Java code, assign the control the ID `stackview`. The value of the `android:animateLayoutChanges` attribute is set to `true` so that changes occurring in the layout will not mandate running `LayoutTransition`.

To represent the stack item that you want to stack in `StackView`, you need to define an XML file in the `res/layout` folder. Right-click the `res/layout` folder in the Package Explorer window, and add an XML file called `item.xml`. Because you want to stack only the images, only an `ImageView` control is defined in the `item.xml` file. After you define the `ImageView`, the `item.xml` file will appear, as shown in Listing 4.6.

Listing 4.6 Code Written in the `item.xml` File

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ImageView
        android:id="@+id/imageview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/ic_launcher" />
</FrameLayout>

```

You can see that the `ImageView` control is assigned the ID `imageview` and is initialized to display the `ic_launcher.png` file. In all, you want to display five images through the `StackView` control. Assuming the five image filenames are `prod1.png`, `prod2.png`, `prod3.png`, `prod4.png`, and `prod5.png`, copy them to the `res/drawable` folders. It's time to write code in the Java activity file to perform the following tasks:

- Access the `StackView` from the layout file and map it to the `StackView` object.
- Define an array to contain the resource IDs of the images that you copied into `res/drawable` folders. This array will act as a data source, providing the images that you want to display.
- Define a custom adapter called `ImageAdapter` that will extend the `BaseAdapter` abstract class to define the content to be displayed through the `StackView` control.
- Display the adapter's content (images) via `StackView`, and set `ImageAdapter` to the `StackView` object via the `setAdapter()` method.

To accomplish all the preceding tasks, write the code as shown in Listing 4.7 in the Java activity file `StackViewAppActivity.java`.

Listing 4.7 Code Written in the Java Activity File `StackViewAppActivity.java`

```
package com.androidtablet.stackviewapp;

import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.StackView;
import android.widget.BaseAdapter;

public class StackViewAppActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stack_view_app);
        StackView stackView = (StackView)this.findViewById(
            R.id.stackview);
        stackView.setAdapter(new ImageAdapter(this));
    }

    public class ImageAdapter extends BaseAdapter {
        private Context context;
        Integer[] images = {
            R.drawable.prod1,
            R.drawable.prod2,
            R.drawable.prod3,
            R.drawable.prod4,
            R.drawable.prod5
        };
    }
};
```

```

public ImageAdapter(Context c) {
    contxt = c;
}

public int getCount() {
    return images.length;
}

public Object getItem(int position) {
    return position;
}

public long getItemId(int position) {
    return position;
}

public View getView(int position, View view, ViewGroup
parent) {
    if (view == null) {
        LayoutInflater vi = (LayoutInflater)
            getBaseContext().getSystemService(
                Context.LAYOUT_INFLATER_SERVICE);
        view = vi.inflate(R.layout.item, null, false);
    }
    ImageView imageView = (ImageView) view.findViewById(
        R.id.imageview);
    imageView.setImageResource(images[position]);
    return view;
}
}
}

```

The `ImageAdapter` is set to the `StackView` control, so it can access the adapter methods to display content (images). The adapter's methods—`getCount()`, `getItem()`, and `getItemId()`—are used to determine the number of images to be displayed and the unique identifier of the specified image. The `getView()` method is used to retrieve the appropriate view or image at the specified position. The `ImageView` defined in the `item.xml` file is accessed and is used to display images through the `StackView`.

After running the application, you find the stack of items (images) (see Figure 4.4 [left]). When you flip the front image, the images in back are moved to the front, as shown in Figure 4.4 [right].



Figure 4.4 StackView displaying images (left), and the hidden images displayed in the front after flipping the front images (right)

The size of the images when an application is run on a phone may appear fine. But on a tablet, the images appear very small. To scale the images according to the screen size of the device, you need to modify the `item.xml` file. Open the `item.xml` file in the `res/layout` folder, and modify it to appear as shown in Listing 4.8. Only the code in bold is the modified code; the rest is the same that you saw in Listing 4.6.

Listing 4.8 Code Written in the `item.xml` File

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ImageView
        android:id="@+id/imageview"
        android:layout_width="@dimen/image_width"
        android:layout_height="@dimen/image_height"
        android:src="@drawable/ic_launcher" />
</FrameLayout>
```

The image(s) that will be displayed through StackView are assigned the width and height through the dimension resources `image_width` and `image_height`, respectively.

To define the dimension resources `image_width` and `image_height`, open the `dimens.xml` file from the `res/values` folder. You assume that the dimension file `dimens.xml` already exists in the `res/values` folder of the application. You also assume that two folders named `values-sw600dp` and `values-sw720dp` exist in the `res` folder, and both the folders contain a dimension file named `dimens.xml`.

To define the width and height for the images when an application runs on a phone, open the `dimens.xml` file in the `res/values` folder, and write the following code in it:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="image_width">100dp</dimen>
    <dimen name="image_height">200dp</dimen>
</resources>
```

You can see that on the phone, the `StackView` will display the images of width and height 100dp and 200dp, respectively.

Again, to define the width and height for the images when an application is viewed on a 7-inch tablet, open the dimension file `dimens.xml` in the `res/values-sw600dp` folder, and write the following code in it:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="image_width">200dp</dimen>
    <dimen name="image_height">300dp</dimen>
</resources>
```

The preceding code will assign 200dp width and 300dp height to the images displayed through `StackView` on a 7-inch tablet. For defining the image dimensions for a 10-inch tablet, open the dimension file `dimens.xml` found in the `res/values-sw720dp` folder, and write the following code in it:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="image_width">300dp</dimen>
    <dimen name="image_height">400dp</dimen>
</resources>
```

The preceding code will make the images appear 300dp wide and 400dp high in `StackView` when an application runs on a 10-inch tablet.

After you run the application on a 10-inch tablet, the `StackView` will appear as shown on the phone in Figure 4.5 (left). Compared to Figure 4.4 (left), you can see that the images appear quite big and clear on a tablet. When you flip a front image, the images at the back move to the front, as shown in Figure 4.5 (right).



Figure 4.5 stackView displaying enlarged images (left); the images at the back appear in the front when you flip the front images (right).

Recipe: Displaying a List of Options Using ListPopupWindow

You can use `ListPopupWindow` to anchor to a host view and display a list of options. In this recipe, you will learn to anchor `ListPopupWindow` to an `EditText` control. When the user clicks in the `EditText` control, `ListPopupWindow` will appear displaying a list of options. After the user selects an option from `ListPopupWindow`, it will be assigned to the `EditText` control. Create a new Android project called `ListPopupWindowApp`.

You want to anchor `ListPopupWindow` to an `EditText` control, so define an `EditText` control in the layout file. After you define an `EditText` control, the activity layout file `activity_list_popup_window_app.xml` will appear as shown in Listing 4.9.

Listing 4.9 Code Written in the Activity Layout File `activity_list_popup_window_app.xml`

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText
```



```

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/product_name"
        android:hint="Enter product name"
        android:textSize="@dimen/text_size" />
</LinearLayout>

```

You can see that the `EditText` control is assigned the ID `product_name`. In this application, you will prompt the user to enter a product name in the `EditText` control. The text `Enter product name` is displayed in the `EditText` control. The text entered into the `EditText` control will appear in the font size defined by the dimension resource `text_size`.

The default size of the list items displayed in a `ListView` is suitable for phones but is quite smaller for tablets. To resize the list items of the `ListView` as per the device screen size, add one more XML file named `list_item.xml` to the `res/layout` folder. Write the code as shown in Listing 4.10 to the `list_item.xml` file.

Listing 4.10 Code Written in the `list_item.xml` File

```

<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="6dp"
        android:textSize="@dimen/text_size"
        android:textStyle="bold" />

```

According to the preceding code, the list items of the `ListView` will be padded by 6dp spaces, will appear in bold, and will be the size defined in the dimension resource `text_size`.

Next, you need to write Java code to perform the following tasks:

- Access the `EditText` control from the layout file and map it to the `EditText` object.
- Define an object of `ListPopupWindow`.
- Define `ArrayAdapter` and set it to the list of products you want to display through `ListPopupWindow`.
- Set `ArrayAdapter` to `ListPopupWindow` to display the list of products defined in `ArrayAdapter`.
- Set the height and width of `ListPopupWindow`.
- Assign a modal nature to `ListPopupWindow` (that is, the control will not return to the caller until the `ListPopupWindow` is dismissed). `ListPopupWindow` will be dismissed either by selecting a product from `ListPopupWindow` or by clicking anywhere outside `ListPopupWindow`.

- Anchor ListPopupWindow to the EditText control.
- Associate setOnItemClickListener to the EditText control so that when the user clicks in the EditText, ListPopupWindow opens and shows the list of products.
- Set the activity class to implement OnItemClickListener. When an option is selected from ListPopupWindow, it is assigned to the EditText control.

To perform these listed tasks, write the code shown in Listing 4.11 to the main Java activity file, ListPopupWindowAppActivity.java.

Listing 4.11 Code Written in the Java Activity File ListPopupWindowAppActivity.java

```
package com.androidtablet.listpopupwindowapp;

import android.os.Bundle;
import android.app.Activity;
import android.widget.ListPopupWindow;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.view.View.OnClickListener;

public class ListPopupWindowAppActivity extends Activity
    implements OnItemClickListener {
    EditText productName;
    ListPopupWindow listPopupWindow;
    String[] products={"Camera", "Laptop", "Watch", "Smartphone",
        "Television"};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_popup_window_app);
        productName = (EditText) findViewById(
            R.id.product_name);
        listPopupWindow = new ListPopupWindow(
            ListPopupWindowAppActivity.this);
        listPopupWindow.setAdapter(new ArrayAdapter(
            ListPopupWindowAppActivity.this,
            R.layout.list_item, products));
        listPopupWindow.setAnchorView(productName);
        listPopupWindow.setWidth(300);
        listPopupWindow.setHeight(400);
```

```

        listPopupWindow.setModal(true);
        listPopupWindow.setOnItemClickListener(
            ListPopupWindowAppActivity.this);
        productName.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                listPopupWindow.show();
            }
        });
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
        productName.setText(products[position]);
        listPopupWindow.dismiss();
    }
}

```

In the preceding code, you see the use of an `ArrayAdapter`, which acts as the data source for `ListPopupWindow`. An `ArrayAdapter` makes use of a `TextView` control to represent the child views in a view (that is, it displays the elements of the `products` array via a `TextView` control). The `ArrayAdapter` constructor used earlier consists of the following:

- **`ListPopupWindowAppActivity.this`**—The current context.
- **`R.layout.list_item`**—Points to the `TextView` that you defined in the `list_item.xml` file. The `TextView` will be used to display each item in `ListPopupWindow`. The elements of the `products` array are wrapped in a view before being assigned to the widget for display. Therefore, the `R.layout.list_item` simply turns the strings defined in the `products` array into a `TextView` for display in `ListPopupWindow`.
- **`products`**—Acts as a data source.

After running the application, you get an `EditText` control with the message `Enter product name` (see Figure 4.6 [(top)]). Click in the `EditText` control, and `ListPopupWindow` appears showing the list of products (see Figure 4.6 [(middle)]). After you select a product from `ListPopupWindow`, it will appear in the `EditText` control (see Figure 4.6 [(bottom)]).

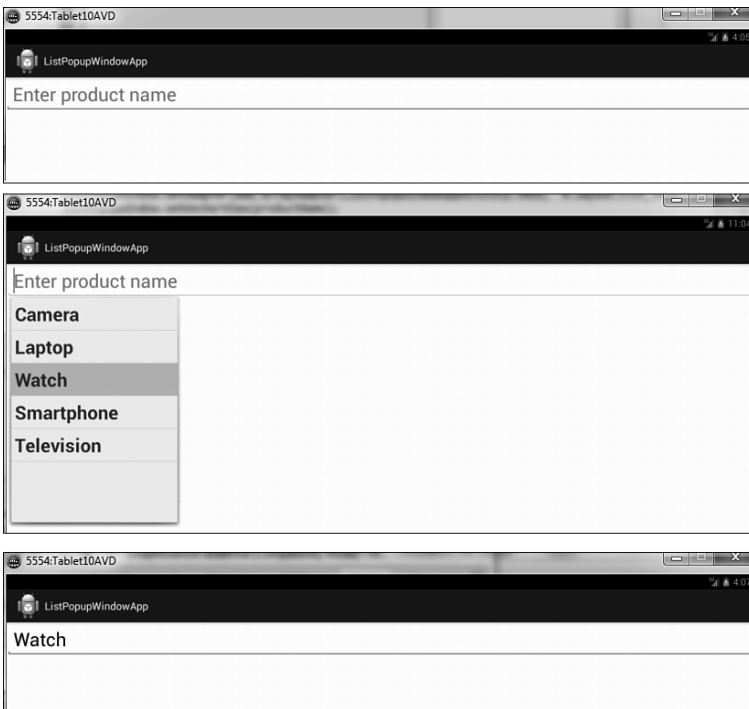


Figure 4.6 `EditText` prompting to enter a product name (top); `ListPopupWindow` appears showing the options after clicking in the `EditText` control (middle); the selected product from `ListPopupWindow` appears in `EditText` (bottom).

Recipe: Suggesting Options Using `PopupMenu`

`PopupMenu` displays a menu in a modal pop-up window. You can anchor it to a view and make it display the desired menu items or options. In this recipe, you will anchor `PopupMenu` to an `EditText` control to display a list of suggestions while entering data in the `EditText` control. The difference between the previous recipe and this one is that the list of options is displayed through `PopupMenu` instead of `ListPopupWindow`.

Create a new Android project called `PopupMenuApp`. Because you want to anchor `PopupMenu` to the `EditText` control, it is defined in the layout file `activity_popup_menu_app.xml` using the code shown in Listing 4.12.

Listing 4.12 Code Written in the Activity Layout File `activity_popup_menu_app.xml`

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/product_name"
        android:hint="Enter product name"
        android:textSize="@dimen/text_size" />
</LinearLayout>

```

You can see that the `EditText` control is assigned the ID `product_name` and is set to display the message `Enter product name`.

You will define the menu items or options for `PopupMenu` through the XML file. In other words, you will inflate the menu through an XML file. To the `res/menu` folder, add an XML file called `popupmenu.xml`. You want to display product names in the form of suggestions in the `EditText` control, so define the menu items in the form of product names in the `popupmenu.xml` file. The menu items are defined as shown in Listing 4.13 in the `popupmenu.xml` file.

Listing 4.13 Code Written in the `popupmenu.xml` File

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:id="@+id/group_popupmenu">
        <item android:id="@+id/camera"
            android:title="Camera"
            android:textSize="@dimen/text_size" />
        <item android:id="@+id/laptop"
            android:title="Laptop"
            android:textSize="@dimen/text_size" />
        <item android:id="@+id/watch"
            android:title="Watch"
            android:textSize="@dimen/text_size" />
        <item android:id="@+id/smartphone"
            android:title="Smartphone"
            android:textSize="@dimen/text_size" />
        <item android:id="@+id/television"
            android:title="Television"
            android:textSize="@dimen/text_size" />
    </group>
</menu>

```

You can see that products, Camera, Laptop, Watch, Smartphone, and Television are defined as menu items in the `popupmenu.xml` file. Each product name is assigned a unique ID, too.

You need to write Java code to accomplish the following tasks:

- Access the `EditText` control defined in the layout file and map it to the `EditText` object.
- Define the `PopupMenu` object and inflate the menu items or product name defined in the `popupmenu.xml` file to be displayed through `PopupMenu`.
- Associate `setOnClickListener` to the `EditText` control to listen for an occurrence of the click event in the `EditText` control.
- Display the `PopupMenu` when the user clicks in the `EditText` control.
- Associate `setOnMenuItemClickListener` to `PopupMenu`.
- When any menu item (product) is selected from `PopupMenu`, it is assigned to the `EditText` control.

To perform the preceding tasks, write the code shown in Listing 4.14 into the main Java activity file, `PopupMenuAppActivity.java`.

Listing 4.14 Code Written in the Java Activity File `PopupMenuAppActivity.java`

```
package com.androidtablet.popupmenuapp;

import android.os.Bundle;
import android.app.Activity;
import android.widget.EditText;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.PopupMenu;
import android.view.MenuItem;

public class PopupMenuAppActivity extends Activity {
    EditText productName;
    PopupMenu popupMenu;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_popup_menu_app);
        productName = (EditText) findViewById(
            R.id.product_name);
        popupMenu = new PopupMenu(PopupMenuAppActivity.this,
            productName);
```

```

popupMenu.getMenuInflater().inflate( R.menu.popupmenu,
    popupMenu.getMenu());
productName.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        popupMenu.setOnMenuItemClickListener(new
            PopupMenu.OnMenuItemClickListener() {
                @Override
                public boolean onMenuItemClick(MenuItem
                    item) {
                    productName.setText(item.toString());
                    return true;
                }
            });
        popupMenu.show();
    }
});
}
}
}
}

```

After you run the application, the `EditText` appears on startup. The `EditText` control displays a message directing the user to enter a product name (see Figure 4.7 [top]). When the user clicks in the `EditText` control, the `PopupMenu` appears showing product names in the form of menu items (see Figure 4.7 [middle]). The user selects a product from the `PopupMenu`, and it is assigned to the `EditText` control (see Figure 4.7 [bottom]).

You see that `PopupMenu` appears below the anchor view (`EditText` control) because there is enough space below the `EditText` control. If there is not enough space, `PopupMenu` would have appeared above the anchor view.

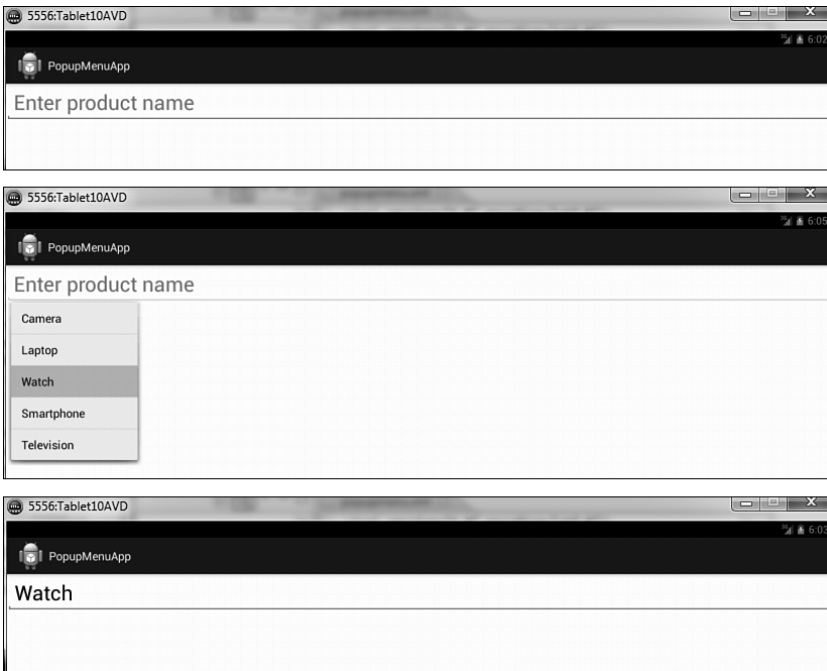


Figure 4.7 `EditText` prompting to enter a product name (top); `PopupMenu` appears showing the options after clicking in the `EditText` control (middle); the selected product from `PopupMenu` appears in the `EditText` (bottom).

Summary

In this chapter, you learned to display the calendar in an Android application through `CalendarView` and saw how a date selected from the calendar is displayed. You also learned to display a range of numbers through `NumberPicker`. You walked through the procedure to display a stack of images using the `StackView` widget. Finally, you learned to display a list of options using `ListPopupWindow` and display suggestions through `PopupMenu`.

The next chapter is focused on understanding `ClipData` and `DragEvent`. You will learn about the system clipboard and the procedure to drag and drop text and images.

This page intentionally left blank

Index

Numerics

10-inch tablets

- defining dimension resources, 168-169
- layouts, 20

7-inch tablets

- defining dimension resources, 168-169
- layouts, 20

A

AccelerateDecelerateInterpolator, 273

AccelerateInterpolator, 273, 276-279

accelerometer sensor, 414-417

- registering listeners, 410

AccessContactApp project, 204-209

accessing

- camera, 323
- Contacts information with
CursorLoader, 204-209
- device sensors, 409-410
- information with JSONObject, 428-430
- JSON data from Web service, 444-446
- online content with WebView widget,
450-451

action items

- adding to ActionBar, 107
- displaying in ActionBar, 104-107
- displaying in drop-down list, 126-129
- representing with icons, 105
- responses to, displaying, 114
- submenus, displaying, 112-116

action views, displaying in ActionBar, 109-112

ActionBar, 101

action items

- adding, 107
- representing with icons, 105
- responses to, displaying, 114

action items, displaying, 104-107

action views, displaying, 109-112

comparing to menus, 101-102

components of, 103

drop-down list ActionBar, creating, 125-129

hiding, 102

logos, navigating application home page, 107-109

menu item clicks, 108

menu items, making checkable, 116

navigation tabs, displaying, 120

radio buttons, 116-119

SearchView widget, 111

submenus, displaying, 112-116

tabbed ActionBar, creating, 120-125

visibility, toggling, 102

ActionBarListApp project, 126-129

ActionBarOnOlderApp project, 470-473

ActionBarSubmenu project, 112-116

ActionBarTabApp project, 120-125

ActionItemsApp project, 105-107

ActionViewApp project, 109-112

activities, 30

creating, 15-16

data, passing to other activities, 40-45

dialogs, displaying through

DialogFragment, 84-90

- fragments
 - background fragments, 51
 - communication, implementing between, 75-79
 - creating at runtime, 68-74
 - foreground fragments, 51
 - ListFragments, 80-84
 - managing with FragmentManager, 59-65
 - PreferenceFragment, setting user preferences with, 90-98
 - removing, 68
 - WebViewFragment, 460-465
- hardware acceleration, applying, 282
- intents, 35-39
 - broadcast receivers, 186
 - methods used for, 185
 - pending intents, 183-185
 - receiving, 189-192
- life cycle, 30-34
- subactivities, adding to
 - AndroidManifest.xml file, 45
- synchronous dialogs, 84
- WelcomeActivity, creating, 37-39
- ActivityLifeCycleApp project, 32-34**
- adapters, ArrayAdapter, 148**
- adapting to screen orientation**
 - by defining alternate layouts, 483-488
 - through anchoring controls, 480-482
- adding**
 - action items to ActionBar, 107
 - Android Support Library, 478
 - fragments to containers, 73-74
 - fragments with device orientation, 59-65
 - subactivities to AndroidManifest.xml file, 45
 - tabs to ActionBar, 120-125
- AlarmManager, updating home screen widgets with, 507-510**
- Allocation Tracker tab (DDMS), 394**
- AlphaAnimation class, 260**
- alternate layouts, defining to handle screen orientation changes, 483-488**
- anchoring controls, 480-482**
- Android 3.0 SDK, 7**
- Android Beam**
 - beaming data between devices, 526
 - transferring data with, 527-531
- Android projects**
 - AccessContactApp, 204-209
 - ActionBarListApp, 126-129
 - ActionBarOnOlderApp, 470-473
 - ActionBarSubmenu, 112-116
 - ActionBarTabApp, 120-125
 - ActionItemsApp, 105-107
 - ActionViewApp, 109-112
 - ActivityLifeCycleApp, 32-34
 - AnimationSetApp, 274-279
 - AudioCaptureApp, 344-349
 - AudioRecordApp, 334-336
 - BlueToothApp, 368-374
 - BTFileTransferApp, 375-378
 - CalendarViewApp, 131-134
 - CamcorderApp, 337-342
 - CameraApp, 323-326
 - CaptureImageApp, 328-333
 - CommunicateDataApp, 41-45
 - CustomContentProviderApp, 209-219
 - directory structure, 14-19
 - DragandDropImage, 166-175
 - DragDropListApp, 157-165
 - ForegroundFragmentApp, 51-59
 - FragmentByCodeApp, 68-74
 - FragmentOnOlderApp, 474-480
 - FrameAnimationApp, 253-257
 - HardwareAccApp, 282-283
 - JSONApp, 428-430
 - JSONArrayApp, 433-437
 - JSONReaderWriterApp, 439-442
 - ListPopupWindowApp, 145-148
 - MultiAnimApp, 248-251
 - MultipleThreadsApp, 398-402
 - NFCApp, 521-526

NFCReadTagApp, 516-521
 NumberPickerApp, 135-139
 ObjectAnimatorApp, 240-247
 OpenGLApp, 300-305
 OpenGLDemo, 309-313
 PendingIntentApp, 196-199
 PopupMenuApp, 149-153
 PrefFragmentApp, 91-98
 SensorProximityApp, 418-420
 SensorsListApp, 411-413
 StackViewApp, 139-144
 SurfaceViewApp, 288-294
 TextureViewApp, 295-297
 TimeHomeWidgetApp, 495-502
 TweeningAnimApp, 258-268
 ValueAnimatorApp, 234-236
 VideoCaptureApp, 354-359
 VideoRecordApp, 350-352
 ViewLayerApp, 284-287
 WebViewFragApp, 460-466
 WiFiApp, 379-383
 WiFiDirectApp, 384-388

Android Support Library, 473-480

animations

collecting with AnimationSet class, 276-279
 constants used by, 233
 duration of, defining, 235-236
 frame-by-frame animations, 232, 252-257
 interpolators, 273
 layout animations, 232, 268-273
 methods used by, 233
 multiple animations, creating with AnimatorSet class, 248-251
 property animations, 231
 sequencing with AnimationSet class, 273-279
 tweening animations, 232, 258-268
 AlphaAnimation class, 260
 RotateAnimation class, 261-263

 ScaleAnimation class, 263-268
 TranslateAnimation class, 261
 view animations, 231-232

AnimationSet class, sequencing animations with, 273-279

AnimationSetApp project, 274-279

animators

ObjectAnimator, 231
 animating views with, 240-247
 ValueAnimator, 231-236
 methods used for, 232

AnimatorSet class, creating multiple animations with, 248-251

AnticipateInterpolator, 273

AnticipateOvershootInterpolator, 273

APIs, requirements for graphics rendering, 299-300

app widget hosts, 491

app widgets, 491-493

available app widgets, displaying, 492-493
 life cycle methods, 493

applications

activities, 30
 creating, 15-16
 life cycle, 30-34
 passing data between, 40-45
 AsyncTaskApp, 403-405
 compatibility
 for different screens, enabling, 467-469
 enabling for phones and tablets, 10
 platform versions, supporting different types of, 468
 improving performance with layers, 284-287
 menus, comparing to ActionBars, 101-102
 phone application, converting to tablet application, 19-29
 resources, 18
 tablet-only compatibility, forcing, 29-30

- tasks, performing with background threads, 395-398
- tracking with Web analytics tool, 533-534
 - EasyTracker library, 534-542
 - GoogleAnalytics singleton, 543-545
- user preferences, setting with PreferenceFragment, 90-98
- using cut, copy, and paste operations between, 176-180

applying

- custom styles to action item dropdown list, 127
- hardware acceleration to activities, 282
- layout animations, 268-273
- smooth coloring, 304-308
- transformations with TextureView, 294-297

ArrayAdapter, 148**arrays, JSONArray, 433-437****assigning icons to notifications, 193****assumptions for recipes, 6****AsyncTask class, 402-405****attributes**

- of frame_anim.xml file, 255
- for LayoutAnimationController class, 268

audio, recording

- with built-in intent, 333-336
- with Java code, 344-349

AudioCaptureApp project, 344-349**AudioRecordApp project, 334-336****available app widgets, listing, 492-493****available sensors for device, listing, 411-413****AVDs (Android Virtual Devices)**

- creating, 10-13
- launching, 13

beaming data between devices, 526**Bluetooth, 361. See also Android Beam; Wi-Fi Direct**

- Bluetooth device, pairing with Windows PC, 366-367
- files, transferring, 375-378
 - between two Bluetooth devices, 364
- local Bluetooth device, enabling, 368-371
- paired devices, displaying, 371-374
 - two Bluetooth devices, pairing, 361-364

BluetoothAdapter class, 368**BluetoothApp project, 368-374****BounceInterpolator, 273****broadcast receivers, 186**

- defining for home screen widgets, 497

broadcasting intents, 185-192

- broadcast receivers, 186
- methods used for, 185

BTFileTransferApp project, 375-378**build files, project.properties, 19****builder classes, Notification.Builder, 194****built-in content providers, 202****built-in intents**

- audio, recording, 333-336
- images, capturing, 323-326
- video, recording, 350-352

Bundle object, passing data between activities, 40-41**Button controls**

- displaying, 36-37
- home screen widgets, updating, 502-506

buttons

- radio buttons, 116-119
- Start Activity, creating, 37-39

B**background fragments, 51****background threads, performing tasks with, 395-398**

C**calendar, displaying, 131-134****CalendarView widget, displaying calendar with, 131-134**

CalendarViewApp project, 131-134**callback methods**

onKey(), 455

onSelectedDayChange(), 134

camcorder, displaying profile settings, 338**CamcorderApp project, 337-342****CamcorderProfile class, 337-342****camera**

accessing, 323

images, capturing

with built-in intent, 323-326

with Java code, 327-333

images, saving to SD card, 332

Camera class, 328**CameraApp project, 323-326****CaptureImageApp project, 328-333****capturing images**

with built-in intent, 323-326

with Java code, 327-333

characteristics of content providers, 202**classes**

AlphaAnimation, 260

AnimationSet, 273-279

AnimatorSet

creating multiple animations with,
248-251

methods used for, 248

AsyncTask, 402-405

BluetoothAdapter, 368

CamcorderProfile, 337-342

Camera, 328

ClipboardManager, 176-180

ClipData, 176

CursorLoader, 201-202

for drag and drop operation, 156

Handler, 395-398

JsonReader, 438-442

JsonWriter, 438-442

MediaRecorder, 343, 353-359

NotificationManager, obtaining, 195

notifications, creating, 192-193

RotateAnimation, 260

parameters, 262

ScaleAnimation, 260

SensorManager, 414

SharedPreferences, 91

SurfaceView, 288-294

TranslateAnimation, 260

ValueAnimator, methods used for, 232

WebViewClient, 457-459

WiFiManager, 379-380

clip, 176**clipboard, copying/cutting data into,
176-180****ClipboardManager class, 176-180****ClipData, 176****collecting animations with AnimationSet
class, 273-279****combining animations with AnimationSet
class, 273-279****CommunicateDataApp project, 41-45****communication, implementing between
fragments, 75-79****comparing**

ActionBars and menus, 101-102

phones and tablets, 9-10

TextureView and SurfaceView, 294-295

compatibility

Android Support Library, 473-480

of applications, enabling for phones
and tablets, 10, 22platform versions, supporting different
types of, 468

screen compatibility, enabling, 467-469

tablet-only compatibility, forcing, 29-30

components of ActionBar, 103**configuring**

EasyTracker library options, 538-539

keyboard settings, 407

notifications, 194-195

constants

for animations, 233

for DragEvent class, 156

- constructors for JSONArray, 433**
- consuming Web services (JSON), 442-446**
- contacts, inserting with emulator, 204-205**
- containers**
 - adding fragments, 73-74
 - LinearLayout, 157
- content providers, 202-204**
 - Contacts, accessing information with CursorLoader, 204-209
 - creating, 209-219
 - fetching data from, 203-204
 - query URI, 203-204
 - rows, inserting, 216-218
- controlling visibility**
 - of logos, 102
 - of titles, 102
- controls, anchoring, 480-482**
- converting**
 - phone applications to tablet applications, 19-29
 - pixels into device-independent units, 9
- copying data into clipboard, 176-180**
- copying, 176-180**
- Create tab, adding to ActionBar, 120-125**
- creating**
 - activities, 15-16
 - WelcomeActivity, 37-39
 - Android projects, 14-19
 - AVDs, 10-13
 - content providers, 209-219
 - DialogFragment, 88-90
 - drag shadows, 163
 - drop-down list ActionBar, 125-129
 - fragments at runtime, 68-74
 - intent objects, 183-184
 - NDEF messages, 513-514
 - notifications, 193
 - rectangles with OpenGL, 300-305
 - tabbed ActionBar, 120-125
 - widgets, home screen widgets, 494-502
- CursorLoader class, 201-202**

- custom content providers**
 - creating, 209-219
 - deleting information from, 223-229
 - listing information in, 219-223
 - product information, adding, 218-219
 - rows, inserting, 216-218
 - updating information in, 223-229
- custom styles, applying to action item drop-down list, 127**
- CustomContentProviderApp project, 209-219**
- customizing WebView widget, 450**
- cut, copy, and paste operations for text, 176-180**
- CycleInterpolator, 273**

D

- data, passing between activities, 40-45**
- DDMS (Dalvik Debug Monitor Service)**
 - Devices tab icons, 393
 - garbage collection, 392-394
- DecelerateInterpolator, 273**
- defineGraphic() method, 303**
- defining**
 - alternate layouts, 483-488
 - dimensions of images, 144
 - icon for new applications, 15
 - ListView control, 51
 - NDEF messages, 527
 - TextView control, 51-52
- deleting**
 - information from custom content provider, 223-229
 - widget instances, 502
- density (screens), enabling compatibility between categories, 467-469**
- device sensors, accessing, 409-410**
- DialogFragment, creating, 88-90**
- dialogs, 84**
- dimension resources, 19**
- dimensions of images, defining, 144**

directory structure of Android projects, 14-19
 disabling hardware acceleration, 282
 discovering available sensors for device, 411-413
 displaying

- action items
 - in drop-down list, 126-129
 - responses to, 114
- action views in ActionBar, 109-112
- ActionBar, 102
 - action items, 104-107
 - navigation tabs, 120
- available app widgets, 492-493
- Button controls, 36-37
- calendar with CalendarView widget, 131-134
- camcorder profile settings, 338
- dialogs through DialogFragment, 84-90
- information in custom content provider, 219-223
- items in fragment from specified source, 80-84
- numbers in a range with NumberPicker widget, 135-139
- paired devices, 371-374
- preference views, 91-93
 - options, 94-95
- PreferenceFragment, 97
- submenus in ActionBar, 112-116
- web pages with WebViewFragment, 460-465
- Wi-Fi, 379-383

dispOption() method, 73-74
downloading code bundle for this book, 6
drag and drop operations, 155-157

- classes used for, 156
- drag shadows, creating, 163
- for images, 166-175
- for text, 157-165

drag shadows, creating, 163
DragandDropImage project, 166-175
DragDropListApp project, 157-165

DragEvent class, constants, 156
drop-down list ActionBar, creating, 125-129
duration of animations, defining, 235-236

E

EasyTracker library

- tracking applications with, 534-542
- tracking options, configuring, 538-539

Eclipse

- Android projects, defining application icon, 15
- AVDs
 - creating, 10-13
 - launching, 13

embedding Google Analytics in applications, 534-536
emulator

- inserting contacts with, 204-205
- launching, 13

enabling

- application compatibility for phones and tablets, 10
- hardware acceleration, 281-282
- local Bluetooth device, 368-371
- screen compatibility, 467-469
- Wi-Fi Direct, 384-388

entering data to JSONObject, 428-430
establishing connection between two Bluetooth-enabled devices, 361-364
event listeners

- TabListener, 125
- unregistering, 410

events, activity life cycle, 30-34
example code for this book, downloading, 6
explicit intents, 35

F

fetching data from content providers, 203-204
fields of NdefRecords, 513

file structure of Android projects, 14-19
files, transferring
 through Bluetooth, 375-378
 between two Bluetooth devices, 364
findFragmentById() method, 67
findFragmentByTag() method, 67
FLAG_ACTIVITY_CLEAR_TOP intent, 109
flags for pending intents, 184
flat color, 304
flipping images with StackView widget, 144
forcing tablet-only compatibility, 29-30
foreground fragments, 51
ForegroundFragmentApp project, 51-59
 fragments, adding with device orientation, 59-65
formulas, converting pixels to device-independent units, 9
FragmentByCodeApp project, 68-74
FragmentManager class, 59-65
FragmentOnOlderApp project, 474-480
fragments
 adding with device orientation, 59-65
 background fragments, 51
 creating dynamically at runtime, 68-74
 displaying items from specified source, 80-84
 foreground fragments, 51
 implementing communication between, 75-79
 life cycle, 48
 ListFragments, 80-84
 ListView control
 defining, 51
 list items, resizing, 52
 PreferenceFragment, 90-98
 displaying, 97
 removing, 68
 TextView control, defining, 51-52
 UI, loading, 52-55
 WebViewFragment, 460-465

FragmentTransaction class, 66-68
frame_anim.xml file, attributes, 255
FrameAnimationApp project, 253-257
frame-by-frame animations, 232, 252-257
functions, setWebViewClient(), 458

G

garbage collection, 392-394
gen folder (Android project), 18
getActionBar() method, 102, 124
getActivity() method, 55, 183-184
getArguments() method, 75
getExtras() method, 41
getLoaderManager() method, 201
getPrimaryClip() method, 176
getStringExtra(), 186
getStringExtra() method, 186
getView() method, 139-144
glClearColor() method, 303
glDrawArrays() method, 303
glScalef() method, 316
Google Analytics SDK, 533
 embedding in applications, 534-536
Google Nexus 10, 7-8
Google Nexus 7, 7-8
GoogleAnalytics singleton, tracking applications with, 543-545
GPU (graphics processing unit), 281
graphics
 APIs required for, 299-300
 rectangles
 applying smooth color to, 304-308
 rendering with OpenGL, 300-305
 rotating, 309-313
 scaling, 315-317
 translating, 318-320
graphics-based performance, improving with SurfaceView class, 288-294
GridView, resizing, 166-175
gyroscope sensor, 420-424

H

Handler class, 395-398

handlers

- for loader callbacks, 202
- multiple threads, implementing, 398-402

hardware acceleration, 281-283

- applying to activities, 282
- disabling, 282
- enabling, 281-282

HardwareAccApp project, 282-283

Heap tab (DDMS), 393

hiding ActionBar, 102

home page of applications, navigating, 107-109

home screen widgets, 491. See also app widgets

- broadcast receivers, defining, 497
- creating, 494-502
- instances of, deleting, 502
- updating, 497
 - with AlarmManager, 507-510
 - with Button control, 502-506

I

icons

- action items, representing with, 105
- for applications, defining in Eclipse, 15
- assigning for notifications, 193
- DDMS Devices tab, 393

images

- capturing
 - with built-in intent, 323-326
 - with Java code, 327-333
- dimensions of, defining, 144
- displaying through StackView widget, 139-144
- drag and drop operations, 166-175
- frame-by-frame animations, 252-257
- saving to SD card, 332

implicit intents, 35

initLoader() method, 201

inserting

- contacts with emulator, 204-205
- rows in content provider, 216-218

intent filters, 516-521

intents

- activities, starting, 35-39
- broadcast receivers, 186
- broadcasting, 185-192
 - methods used for, 185
- built-in intents
 - audio, recording, 333-336
 - images, capturing, 323-326
 - video, recording, 350-352
- explicit intents, 35
- FLAG_ACTIVITY_CLEAR_TOP, 109
- pending intents, 183-185
 - activities, launching on occurrence of notification, 196-199
 - flags, 184
 - receiving, 189-192

interpolators, 273

invalidate() method, 234

J

Java code

- audio, recording, 344-349
- images, capturing, 327-333
- video, recording, 353-359

JSON (JavaScript Object Notation), 425-428. See also JSONObject; JSONArray

- key-value pairs, 425
- Web services, consuming, 442-446

JSONApp project, 428-430

JSONArray, 433-437

- constructors, 433
- methods used by, 433-434

JSONArrayApp project, 433-437

JSONObject, 425

- accessing information with, 428-430
- entering data into, 428-430
- methods, 426
- nesting, 425-426, 430-432

JsonReader class, 438-442
JSONReaderWriterApp project, 439-442
JsonWriter class, 438-442

K-L

keyboard settings, configuring, 407
key-value pairs (JSON), 425
landscape orientation
 adding with fragments, 59-65
 layouts, 22-29
launching
 activities
 on occurrence of notification, 196-199
 with Start Activity button, 37-39
 AVDs, 13
layers
 improving performance with, 284-287
 properties, modifying, 285
layout animations, 232, 268-273
 interpolators, 273
LayoutAnimationController class, attributes, 268
layouts
 alternate layouts, defining to handle screen orientation changes, 483-488
 for tablets, 19
life cycle
 of activities, 30-34
 of fragments, 48
life cycle methods (app widgets), 493
LinearInterpolator, 273
LinearLayout containers, 157
linked web pages, opening with WebViewClient class, 457-459
listeners, registering for accelerometer sensor, 410, 414
ListFragments, 80-84
listing
 available sensors for device, 411-413
 information in custom content provider, 219-223
ListPopupWindow widget, 145-148

ListPopupWindowApp project, 145-148
ListView control
 defining, 51
 layout animations, applying, 269-273
 list items, resizing, 52
loaders, 201-202
 callbacks, 202
 CursorLoader, 201-202
 accessing information from Contacts content provider, 204-209
loading
 fragment UI, 52-55
 preference views, 93-94
 UI in fragments, 52-55
local Bluetooth device, enabling, 368-371
log messages of activity life cycle methods, 34
LogCat window, log messages for activity life cycle methods, 34
logos
 controlling visibility of, 102
 home page, navigating, 107-109
 menu item clicks, 108

M

maintenance phase of custom content provider, 223-229
managing
 fragments with FragmentManager, 59-65
 threads with AsyncTask class, 402-405
measuring rotation with gyroscope sensor, 420-424
MediaRecorder class, 343, 353-359
memory management, garbage collection, 392-394
menu items (ActionBar)
 clicks, 108
 making checkable, 116
 radio buttons, 116-119
 submenus, displaying, 112-116
menus, comparing to ActionBars, 101-102

messages

broadcasted intents, receiving, 189-192

NDEF messages, creating, 513-514

notifications

 configuring, 194-195

 creating, 193

 icons, assigning, 193

 sending with intents, 185-192

methods, 186

activity life cycle, 30-34

for animations, 233

for AnimatorSet class, 248

for BluetoothAdapter class, 368

callback methods, fragment life cycle, 48

defineGraphic(), 303

dispOption(), 73-74

findFragmentById(), 67

findFragmentByTag(), 67

getActionBar(), 102, 124

getActivity(), 55, 183-184

getExtras(), 41

getLoaderManager(), 201

getPrimaryClip(), 176

getView(), 139-144

glClearColor(), 303

glDrawArrays(), 303

glScalef(), 316

initLoader(), 201

invalidate(), 234

for JSONArray, 434

for JSONObject, 426

for JsonReader class, 438

for JsonWriter class, 439

life cycle methods (app widgets), 493

for MediaRecorder class, 343

newIntent(), 176

newPlainText(), 176

newUri(), 176

for Notificaton.Builder, 194

ofFloat(), 243

onActivityCreated(), 49

onAttach(), 49

onCreate(), 31, 49

onCreateDialog(), 88

onCreateView(), 49

onDestroy(), 31

onDestroyView(), 49

onDetach(), 49

onDragEvent(), 155

onDrawFrame(), 303

onDrawShadow(), 164

onItemClick(), 63

onItemLongClick(), 163

onNavigationItemSelected(), 127

onOptionSelected(), 57

onOptionsItemSelected(), 115

onPause(), 31, 49

onResume(), 31, 49

onSelectedDayChange(), 134

onStart(), 31, 49

onStop(), 31, 49

onSurfaceChanged(), 303

onSurfaceCreated(), 302

onTabUnselected(), 125

for pending intents, 184

play(), 248

playSequentially(), 248

playTogether(), 248

post(), 396

putExtra(), 185

query(), 227

sendBroadcast(), 185

for sending intents, 185

setAction(), 185

setDisplayShowHomeEnabled(), 102

setLayerType(), 285

setListAdapter(), 82

setPrimaryClip(), 176

setWrapSelectorWheel(), 137

shouldOverrideUrlLoading(), 457

startActivity(), 36

startAnimation(), 273

startDrag(), 156

- for ValueAnimator class, 232
- for WebView class, 451
- for WiFiManager class, 379-380

MultiAnimApp project, 248-251

multicore processor architecture, 391-392

multiple animations, creating with AnimatorSet class, 248-251

multiple threads, implementing, 398-402

MultipleThreadsApp project, 398-402

N

navigating application home page, 107-109

navigation tabs, displaying in ActionBar, 120

NDEF (NFC data exchange format) messages

- creating, 513-514
- defining, 527
- receiving, 527-528

NdefRecord fields, 513

nesting JSONObject, 425-426, 430-432

newIntent() method, 176

newPlainText() method, 176

newUri() method, 176

NFC (Near Field Communications), 511-512. See also NFC tags

NFC stickers, 512

NFC tags, 512

- intent filters, 516-521
- NdefRecord fields, 513
- reading from, 515-521
- writing to, 521-526

NFCApp project, 521-526

NFCReadTagApp project, 516-521

notification area, 192

Notification.Builder, methods used by, 194

NotificationManager class, obtaining, 195

notifications

- activities, launching on occurrence of, 196-199
- classes used for, 192-193
- configuring, 194-195

creating, 193

icons, assigning, 193

NumberPicker widget, displaying numbers in a specified range, 135-139

NumberPickerApp project, 135-139

numbers, displaying with NumberPicker widget, 135-139

O

ObjectAnimator, 231

animating views with, 240-247

ObjectAnimatorApp project, 240-247

objects

animating, 237-240

Bundle object, passing data between activities, 40-41

ClipData, 176

intent objects, creating, 183-184

properties for animations, 234

WebView, obtaining, 449

obtaining

NotificationManager class, 195

WebView objects, 449

ofFloat() method, 243

older versions of Android, supporting, 473-480

onActivityCreated() method, 49

onAttach() method, 49

onCreate() method, 31, 49

onCreateDialog() method, 88

onCreateView() method, 49

onDestroy() method, 31

onDestroyView() method, 49

onDetach() method, 49

onDragEvent() method, 155

onDrawFrame() method, 303

onDrawShadow() method, 164

onItemClick () method, 63

onItemLongClick() method, 163

onKey() method, 455

online content, accessing with WebView widget, 450-451

onNavigationItemSelected() method, 127

onOptionSelected() method, 57
onOptionsItemSelected() method, 115
onPause() method, 31, 49
onReceive() method, 186
onResume() method, 31, 49
onSelectedDayChange() method, 134
onStart() method, 31, 49
onStop() method, 31, 49
onSurfaceChanged() method, 303
onSurfaceCreated() method, 302
onTabUnselected() method, 125
OpenGL (Open Graphics Library), 299-300
 rectangles, rendering with OpenGL,
 300-305
OpenGLApp project, 300-305
OpenGLDemo project, 309-313
opening
 linked web pages with WebViewClient
 class, 457-459
 notification drawer, 192
options
 displaying list of with
 ListPopupWindow widget, 145-148
 for EasyTracker library, configuring,
 538-539
 suggesting with PopupMenu widget,
 149-153
orientation of screen
 fragments, adding/removing with,
 59-65
 handling
 by defining alternate layouts,
 483-488
 through anchoring controls,
 480-482

P

paired devices, displaying, 371-374
pairing
 Bluetooth device with Windows PC,
 366-367
 two Bluetooth devices, 361-364
parameters for RotateAnimation class, 262

passing data
 between activities, 40-45
 between fragments, 75-79
paste operations, 176-180
pending intents, 183-185
 activities
 launching on occurrence of notifi-
 cation, 196-199
 starting, 184
 flags, 184
 methods used for, 184
PendingIntentApp project, 196-199
performance
 hardware acceleration, 281-283
 disabling, 282
 enabling, 281-282
 improving
 with SurfaceView class, 288-294
 with view layers, 284-287
**performing tasks with background threads,
 395-398**
phones
 applications, converting to tablet appli-
 cation, 19-29
 AVDs
 creating, 10-13
 launching, 13
 comparing with tables, 9-10
 compatibility between, enabling for
 applications, 10
 screen sizes, 9
pixels
 converting to device-independent
 units, 9
 density, enabling compatibility between
 categories, 467-469
plain text, writing to NFC tags, 521-526
**platform versions, supporting different
 types of, 468**
play() method, 248
playSequentially() method, 248
playTogether() method, 248
PopupMenu widget, 149-153

- PopupMenuApp project, 149-153**
- portrait orientation, layouts, 22-29**
- post() method, 396**
- preference views**
 - displaying, 91-93
 - loading, 93-94
 - options, displaying, 94-95
- PreferenceFragment**
 - displaying, 97
 - user preferences, setting with, 90-98
- PrefFragmentApp project, 91-98**
- processors**
 - multicore, 391-392
 - threads
 - managing with AsyncTask class, 402-405
 - multiple threads, implementing, 398-402
 - performing tasks with background threads, 395-398
- product information, adding to custom content provider, 218-219**
- profile settings for camcorder, displaying, 338**
- ProGuard, 19**
- project.properties build file, 19**
- projects. See Android projects**
- properties**
 - animating
 - with ObjectAnimator, 240-247
 - with ValueAnimator, 232-236
 - for animations, 234
 - of layers, modifying, 285
- property animations, 231**
- proximity sensor, 417-420**
- putExtra() method, 185**

Q-R

- query() method, 227**
- query URI (content providers), 203-204**
 - tags, 203
- radio buttons, 116-119**

- rate of change in animations, defining with interpolators, 273**
- reading from NFC tags, 515-521**
- receiving**
 - broadcasted intents, 189-192
 - NDEF messages, 527-528
- recipes, assumptions, 6**
- recording**
 - audio
 - with built-in intent, 333-336
 - with Java code, 344-349
 - video
 - with built-in intent, 350-352
 - with Java code, 353-359
- rectangles, applying smooth color to, 304-308**
- recycling allocated memory with garbage collection, 392-394**
- registering listeners for accelerometer sensor, 410, 414**
- removing**
 - fragments, 68
 - information from custom content provider, 223-229
- rendering graphics**
 - APIs required for, 299-300
 - rectangles, 300-305
- representing action items with icons, 105**
- resizing**
 - GridView, 168-169
 - ListView control items, 52
- resolution, 468**
- resources, 18**
 - dimension resources, defining for tables, 168-169
- responses to action items, displaying, 114**
- RFID (radio frequency identification), 511**
- RotateAnimation class, 260-263**
 - parameters, 262
- rotating graphics, 309-313**
- rotation, measuring with gyroscope sensor, 420-424**
- rows, inserting in content provider, 216-218**

S

- saving images to SD card, 332
- ScaleAnimation class, 260, 263-268
- scaling graphics, 315-317
- screens
 - layouts, 19
 - orientation
 - fragments, adding/removing with, 59-65
 - handling by anchoring controls, 480-482
 - handling by defining alternate layouts, 483-488
 - sizes of, 9
 - supporting different types of, 467-469
- SD card, saving images to, 332
- SearchView widget, 111
- selector wheel, displaying numbers in, 137-139
- sendBroadcast() method, 185
- sending
 - EasyTracker data to Google Analytics, 537
 - messages with intents, 185-192
 - text messages between fragments, 75-79
- SensorManager class, 414
- SensorProximityApp project, 418-420
- sensors, 408-411
 - accelerometer sensor, 410, 414-417
 - available sensors for device, listing, 411-413
 - gyroscope sensor, 420-424
 - proximity sensor, 417-420
- SensorsListApp project, 411-413
- sequencing with AnimationSet class, 273-279
- setAction() method, 185
- setArguments() method, 75
- setDisplayHomeAsUpEnabled() method, 102
- setLayerType() method, 285
- setListAdapter() method, 82
- setOnQueryTextListener event listener, 110
- setPrimaryClip() method, 176
- setting user preferences with PreferenceFragment, 90-98
- setWebViewClient() function, 458
- setWrapSelectorWheel() method, 137
- SharedPreferences class, 91
- shouldOverrideUrlLoading() method, 457
- showAsAction attribute, 104-105
- ShowProductActivity Java activity, 219-223
- sizes of screens, 9
 - compatibility for different screens, enabling, 468
- smooth coloring, applying, 304-308
- specifying duration of animations, 235-236
- stacking images, 139-144
- StackView widget, 139-144
- StackViewApp project, 139-144
- Start Activity button, creating, 37-39
- startActivity() method, 36
- startAnimation() method, 273
- startDrag() method, 156
- starting activities, 35-39
 - with pending intents, 184
- statistics of applications, tracking, 533-534
 - with EasyTracker library, 534-542
 - with GoogleAnalytics singleton, 543-545
- storing data with JSONArray, 433-437
- string resources, 19
- subactivities, adding to AndroidManifest.xml file, 45
- submenus, displaying in ActionBar, 112-116
- suggesting options with PopupMenu widget, 149-153
- supporting different types of screens, 467-469
- SurfaceView class
 - comparing with TextureView, 294-295
 - improving graphics-based performance with, 288-294
- SurfaceViewApp project, 288-294
- synchronous dialogs, 84

syntax for query URI, 203
system bar, notification area, 192

T

tabbed ActionBar, creating, 120-125
tablet-only compatibility, forcing, 29-30
tablets
 applications, creating from phone applications, 19-29
 AVDs
 creating, 10-13
 launching, 13
 compatibility, enabling for applications, 10
 dimension resources, defining, 168-169
 Google Nexus 10, 7-8
 Google Nexus 7, 7-8
 layouts, 19
 phones, comparing with, 9-10
 screen sizes, 9
TabListener event listener, 125
tags for query URIs, 203
tasks, performing with background threads, 395-398
text
 cut, copy, and paste operations, 176-180
 drag and drop operations, 157-165
 writing to NFC tags, 521-526
text messages, sending between fragments, 75-79
TextureView
 applying transformations with, 294-297
 comparing with SurfaceView, 294-295
TextureViewApp project, 295-297
TextView control, defining, 51-52
Theme.Holo theme, 101-102
ThreadApp application, 396-398
threads, 395-398
 background threads, performing tasks with, 395-398
 managing with AsyncTask class, 402-405

multiple threads, implementing, 398-402

Threads tab (DDMS), 393
TimeHomeWidgetApp project, 495-502
titles in ActionBar, controlling visibility of, 102
TNF (Type Name Format), 513
tooggling ActionBar visibility, 102
tracking applications, 533-534
 with EasyTracker library, 534-542
 with GoogleAnalytics singleton, 543-545
transferring data
 with Android Beam, 527-531
 through Bluetooth, 375-376
 between two Bluetooth devices, 364
transformations, applying with TextureView, 294-297
TranslateAnimation class, 260-261
translating graphics, 318-320
turning on hardware acceleration, 281-282
tweening animations, 232, 258-268
 AlphaAnimation class, 260
 RotateAnimation class, 261-263
 ScaleAnimation class, 263-268
 TranslateAnimation class, 261
TweeningAnimApp project, 258-268

U

UI (user interface), loading to fragments, 52-55
unregistering event listeners, 410
Update tab, adding to ActionBar, 120-125
updating
 home screen widgets, 497
 with AlarmManager, 507-510
 with Button control, 502-506
 information in custom content provider, 223-229
URIs (uniform resource identifiers), query URI, 203-204

V

value resources, 19

ValueAnimator, 231-236

ValueAnimatorApp project, 234-236

versions of Android, platform support

different platforms, 469-473

older versions, 473-480

video, recording

with built-in intent, 350-352

with Java code, 353-359

VideoCaptureApp project, 354-359

VideoRecordApp project, 350-352

view animations, 231-232

view layers

improving performance with, 284-287

properties, modifying, 285

ViewLayerApp project, 284-287

views

animating with ObjectAnimator,
240-247

cut, copy, and paste operations,
176-180

drag and drop operations, 155-157

GridView, resizing, 166-175

organizing, 48

preference views, displaying, 91-93

visibility

of ActionBar, toggling, 102

of titles, controlling, 102

W-X-Y-Z

web pages

displaying

with WebView widget, 450-457

with WebViewFragment, 460-465

linked web pages, opening with
WebViewClient class, 457-459

Web services (JSON), consuming, 442-446

WebView widget, 449-451

customizing, 450

objects, obtaining, 449

web pages, displaying, 450-457

WebViewClient class, 457-459

WebViewFragApp project, 460-466

WebViewFragment, 460-465

WelcomeActivity, creating, 37-39

widgets

app widgets, 491-493

available app widgets, displaying,
492-493

CalendarView, displaying calendar
with, 131-134

home screen widgets

broadcast receivers, defining, 497

creating, 494-502

instances of, deleting, 502

updating with AlarmManager,
507-510

updating with Button control,
502-506

ListPopupWindow, 145-148

NumberPicker, displaying numbers in a
specified range, 135-139

PopupMenu, 149-153

SearchView, 111

StackView, 139-144

WebView, 449-451

customizing, 450

objects, obtaining, 449

web pages, displaying, 450-457

width of available screens, 9

Wi-Fi, 378-380

enabling, 379-383

Wi-Fi Direct, 384-388

enabling, 384-388

WiFiApp project, 379-383

WiFiDirectApp project, 384-388

WiFiManager class, 379-380

**Windows PC, pairing with Bluetooth device,
366-367**

writing to NFC tags, 521-526