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

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

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 Gratner

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

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`, nest `JSONObjects`, 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.

4

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).

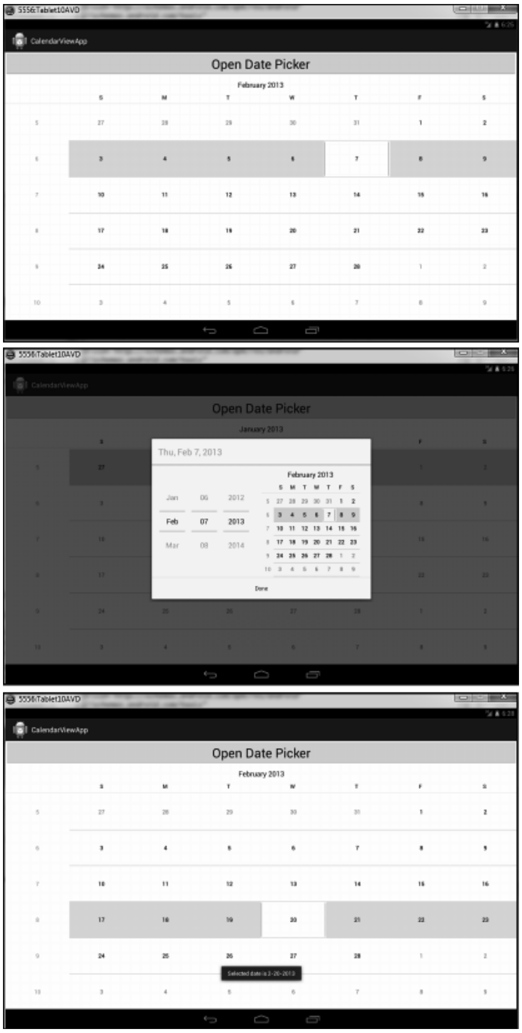


Figure 4.1 `CalendarView` showing the calendar of the current month (top); `DatePickerDialog` opens after selecting the Open Date Picker button (middle); `CalendarView` displays the calendar of the date selected from `DatePicker` (bottom).

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`

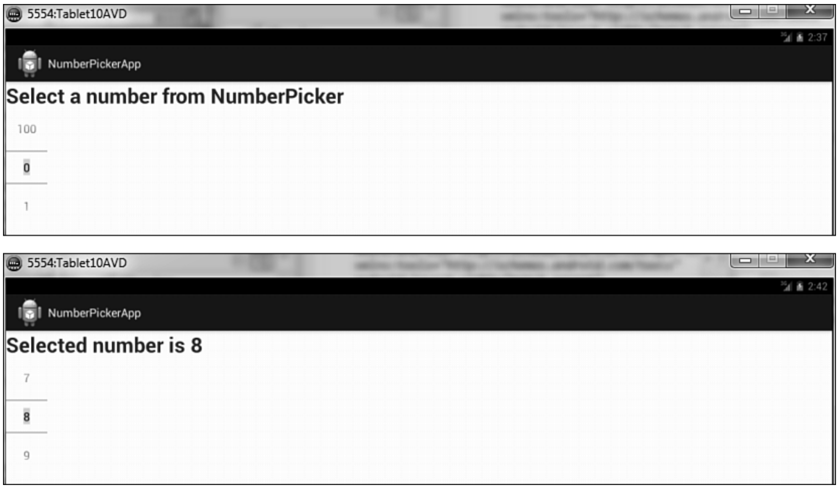
```
package com.androidtablet.numberpickerapp;

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.
            OnValueChangeListener() {
            @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).



Select a number from NumberPicker

Selected number is 8

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:

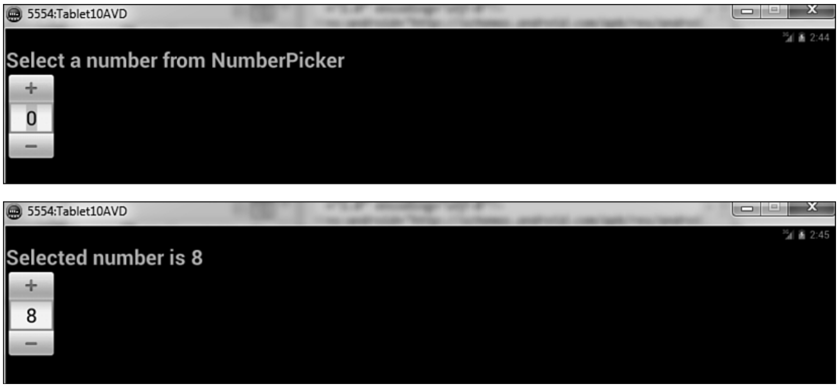
```
String[] stringArray = new String[10];
int n=1;
for(int i=0; i<10; i++){
    stringArray[i] = Integer.toString(n);
    n+=2;
}
numberPicker.setMaxValue(stringArray.length-1);
numberPicker.setMinValue(0);
numberPicker.setDisplayedValues(stringArray);
```

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).


5554:Tablet10AVD

Select a number from NumberPicker

0

5554:Tablet10AVD

Selected number is 8

8

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 contxt;
        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]).

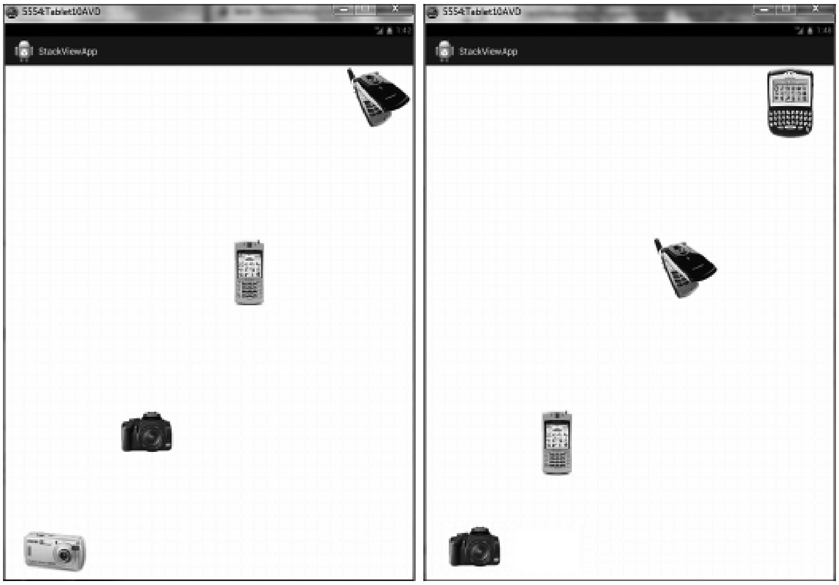


Figure 4.4 `StackView` displaying images (left), and the hidden images displayed in the front after flipping the front images (right)

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).

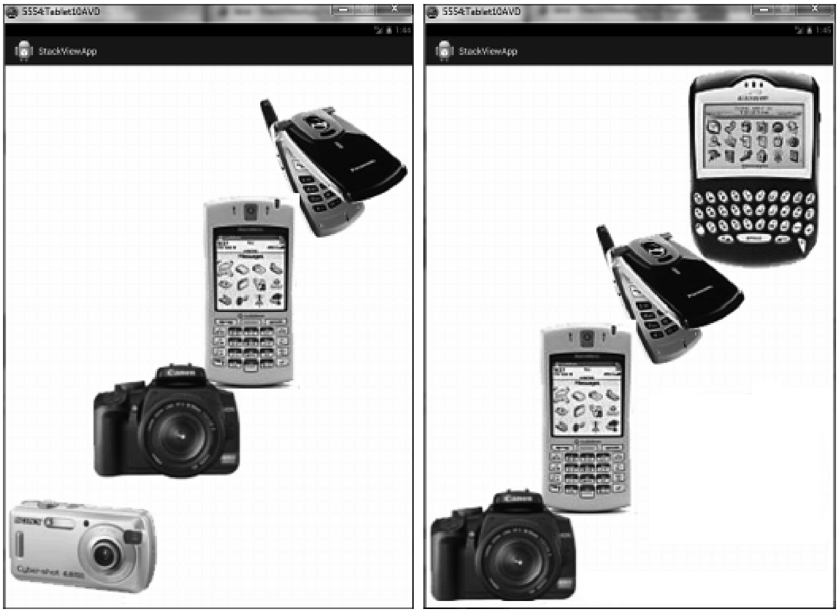


Figure 4.5 StackView displaying enlarged images (left); the images at the back appear in the front when you flip the front images (right).

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.ArrayAdapter;
import android.widget.EditText;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView;
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]).

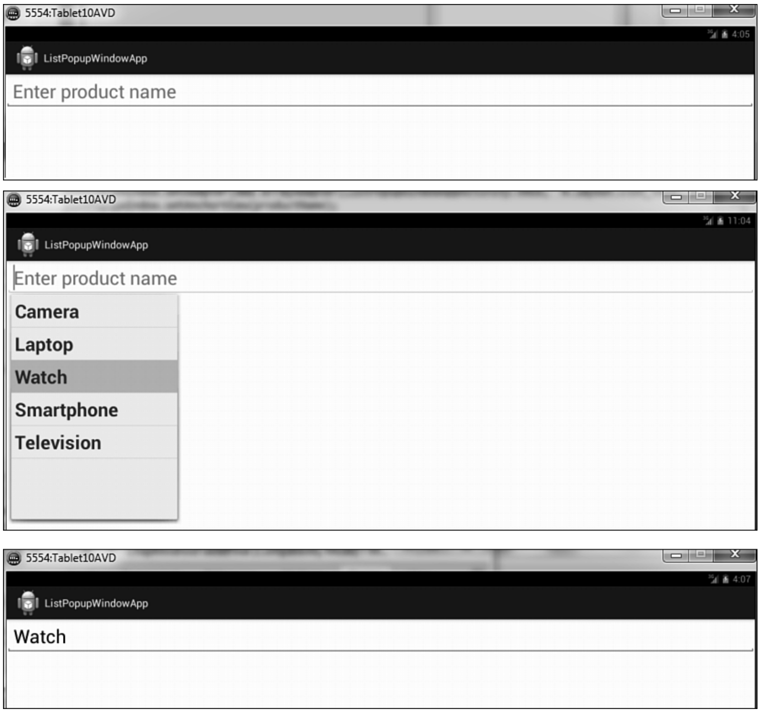


Figure 4.6 `EditText` prompting to enter a product name (top); `ListPopupWindow` appears showing the options after clicking in the `EditText` control (middle); the selected product from `ListPopupWindow` appears in `EditText` (bottom).

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`

```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.

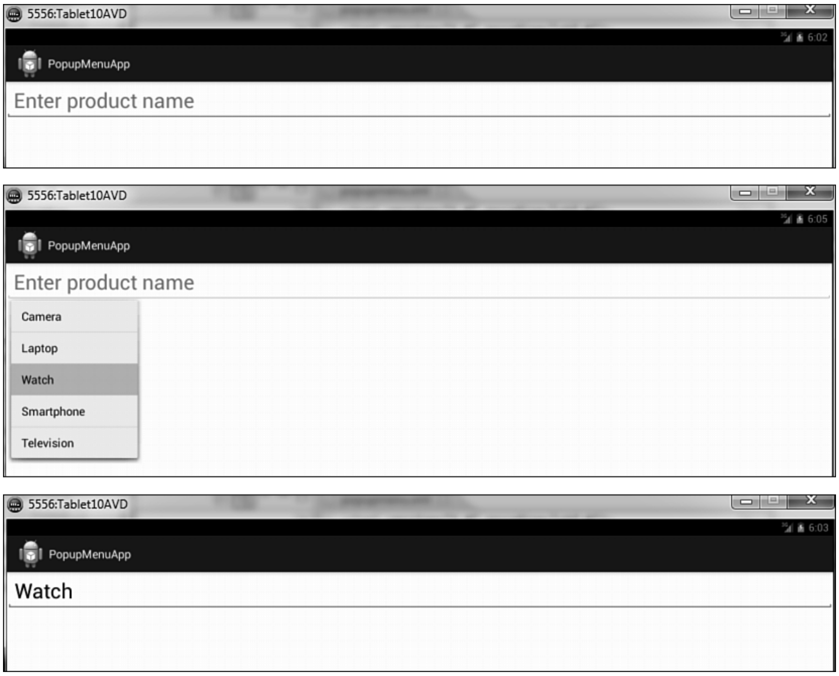


Figure 4.7 `EditText` prompting to enter a product name (top); `PopupMenu` appears showing the options after clicking in the `EditText` control (middle); the selected product from `PopupMenu` appears in the `EditText` (bottom).

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