UNIT 2

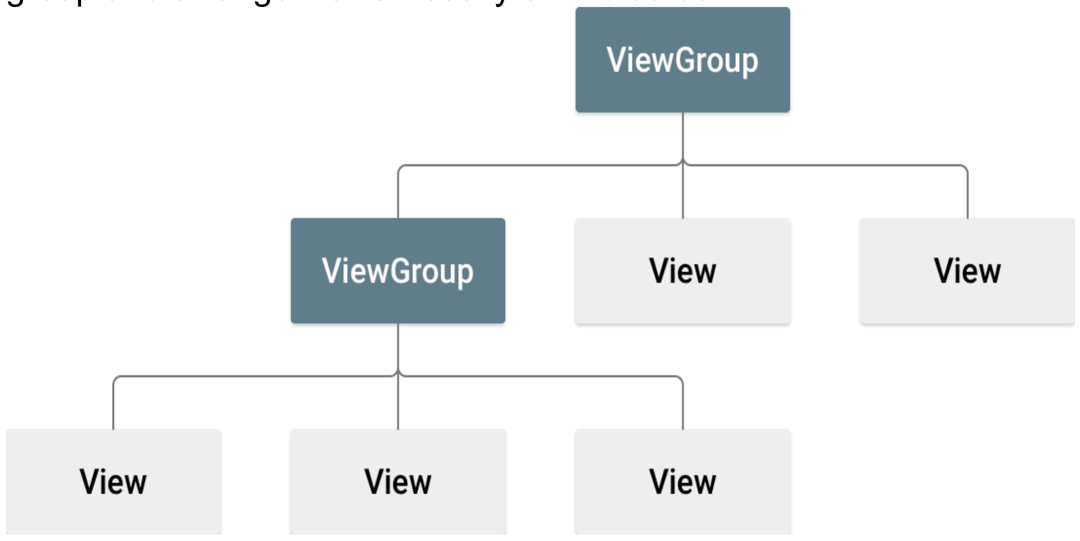
Understanding the Components of a screen

- The basic unit of an Android application is an *activity*, which displays the UI of your application.
- you define your UI using an XML file (for example, the activity_main.xml file located in the res/layout folder of your project)
- During runtime, you load the XML UI in the onCreate() method handler in your Activity class, using the setContentView() method of the Activity class:

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.main); }
```

Views and Viewgroups

- An activity contains *views* and *ViewGroups*.
- A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and text boxes.
- A view derives from the base class **android.view.View.**
- One or more views can be grouped into a ViewGroup.
- A ViewGroup (which is itself a special type of view) provides the layout in which you can order the appearance and sequence of views.
- Examples of ViewGroups include RadioGroup and ScrollView. A ViewGroup derives from the base class **android.view.ViewGroup.**
- Another type of ViewGroup is a Layout
- A Layout is another container that derives from **android .view.ViewGroup** and is used as a container for other views.
- However, whereas the purpose of a ViewGroup is to group views logically—such as a group of buttons with a similar purpose—a Layout is used to group and arrange views visually on the screen.



Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the view.
2	**android:layout_width** This is the width of the layout.
3	**android:layout_height** This is the height of the layout
4	**android:layout_marginTop** This is the extra space on the top side of the layout.
5	**android:layout_marginBottom** This is the extra space on the bottom side of the layout.
6	**android:layout_marginLeft** This is the extra space on the left side of the layout.
7	**android:layout_marginRight** This is the extra space on the right side of the layout.
8	**android:layout_gravity** This specifies how child Views are positioned from top bottom left and write from the window
9	**android:layout_weight** This specifies how much of the extra space in the layout should be allocated to the View.It distributes the available space among the child views
10	**android:layout_x** This specifies the x-coordinate of the layout.
11	**android:layout_y** This specifies the y-coordinate of the layout.
12	**android:layout_width** This is the width of the layout.
13	**android:paddingLeft** This is the left padding filled for the layout.
14	**android:paddingRight** This is the right padding filled for the layout.
15	**android:paddingTop** This is the top padding filled for the layout.

16	**android:paddingBottom** This is the bottom padding filled for the layout.

- The Layouts available to you in Android are as follows:
  - ConstraintLayout
  - FrameLayout
  - LinearLayout (Horizontal) and LinearLayout (Vertical)
  - TableLayout
  - RelativeLayout
  - ScrollView
  - Grid view

ConstraintLayout

- ConstraintLayout is a ViewGroup subclass,Whenever you open the android studio framework the application will be present in the constraint layout.
- In this we have to set the constraint in all four sides.
- This is the default layout.

Framelayout

- FrameLayout is a ViewGroup subclass, The FrameLayout is the most basic of the Android layouts. FrameLayouts are built to hold one view.
- You can add multiple views to a FrameLayout, but each is stacked on top of the previous one. This is when you want to animate a series of images, with only one visible at a time.

Linearlayout (horizontal) and linearlayout (Vertical)

- LinearLayout is a ViewGroup subclass,The LinearLayout arranges views in a single column or a single row. Child views can be arranged either horizontally or vertically, which explains the need for two different layouts—one for horizontal rows of views and one for vertical columns of views.
- LinearLayout(Horizontal) *and* LinearLayout(Vertical)
- *the* android:orientation *property of the* LinearLayout *controls if the application has a horizontal or vertical flow.*

Tablelayout

- TableLayout is a ViewGroup subclass,The TableLayout Layout groups views into rows and columns. You use the <TableRow> element to designate a row in the table. Each row can contain one or more views.
- Each view you place within a row forms a cell. The width of each column is determined by the largest width of each cell in that column.

Relativelayout

- RelativeLayout is a ViewGroup subclass,The RelativeLayout layout enables you to specify how child views are positioned relative to each other.
- each view embedded within the RelativeLayout has attributes that enable it to align with another view.
- These attributes are as follows:

  layout_alignParentTop
  layout_alignParentStart

layout_alignStart
layout_alignEnd
layout_below
layout_centerHorizontal

ScrollView

- A ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display.
- The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout.

GridView

- GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

ListView

- ListView is a view group that displays a list of scrollable items.

- ✓ **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
  This attribute tells the view to size itself to fit the content of its children. For example, if you have a TextView with wrap_content, the view will be sized to fit the text it contains.
- ✓ **android:layout_width=fill_parent** tells your view to become as big as its parent view.
  This attribute instructs the view to expand to fill the entire available space in its parent layout.
  When you set a view's width or height to match_parent, it will expand to fill the available space in its parent.

These attributes are important for controlling the layout and appearance of views within your Android app's user interface.

Common Attributes Used in Views and ViewGroups

attribute	**description**
layout_width	Specifies the width of the view or ViewGroup
layout_height	Specifies the height of the view or ViewGroup
layout_marginTop	Specifies extra space on the top side of the view or ViewGroup
layout_marginBottom	Specifies extra space on the bottom side of the view or ViewGroup
layout_marginLeft	Specifies extra space on the left side of the view or ViewGroup
layout_marginRight	Specifies extra space on the right side of the view or ViewGroup
layout_gravity	Specifies how child views are positioned
layout_weight	Specifies how much of the extra space in the layout should be allocated to the view
layout_x	Specifies the x-coordinate of the view or ViewGroup
layout_y	Specifies the y-coordinate of the view or ViewGroup

Adapting to Display Orientation

- ➢ One of the key features of modern smartphones is their ability to switch screen orientation, and Android is no exception.
- ➢ Android supports two screen orientations: *portrait* and *landscape*.
- ➢ By default, when you change the display orientation of your Android device, the current activity automatically redraws its content in the new orientation.
- ➢ This is because the onCreate() method of the activity is fired whenever there is a change in display orientation.
- ➢ **Note:** When you change the orientation of your Android device, your current activity is actually destroyed and then re-created.
- ➢ when the views are redrawn, they may be drawn in their original locations (depending on the layout selected).

In general, you can employ two techniques to handle changes in screen orientation:

Anchoring—The easiest way is to "anchor" your views to the four edges of the screen. When the screen orientation changes, the views can anchor neatly to the edges.

Resizing and repositioning—Whereas anchoring and centralizing are simple techniques to ensure that views can handle changes in screen orientation, the ultimate technique is resizing each and every view according to the current screen orientation

Managing Changes to screen orientation

Creating an application that displays message base on the screen orientation

Create a Project

App → manifests

→ AndroidManifest.xml

Inside <activity>

```
<activity
        android:name=".MainActivity"
     android:configChanges="orientation|screenSize"     //INCLUDE THIS LINE
         android:exported="true">
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
```

Inside MainActivity.java

\\Enter Ctrl+O select onConfigurationChanged

The following will be displayed

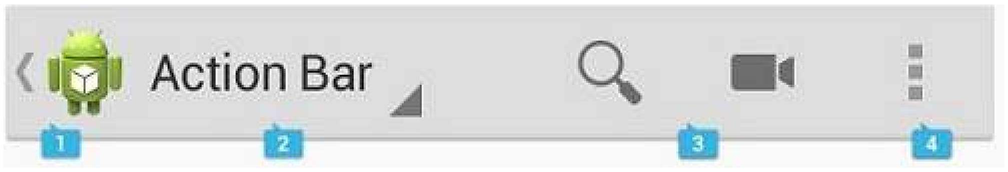
```
@Override
   public void onConfigurationChanged(@NonNull Configuration newConfig) {
      super.onConfigurationChanged(newConfig);
 \\ Include the code which is in BOLD
    If (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
Toast
toast=Toast.makeText(this,"orientation_landscape",Toast.LENGTH_SHORT);
         toast.show();
      }
      else if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT) {
   Toast toast=Toast.makeText(this,"orientation_Portrait",Toast.LENGTH_SHORT);
         toast.show();
      }
   }
```

RUN THE PROGRAM

Utilizing the action bar

- In Android applications, **ActionBar** is the element present at the top of the activity screen.
- Besides fragments, another feature of Android is the Action Bar.
- In place of the traditional title bar located at the top of the device's screen, the Action Bar displays the application icon and the activity title.
- Optionally, on the right side of the Action Bar are action items.
- The setSupportActionBar() method writes your Action Bar to the screen. The Action Bar can be an instance of a Tool Bar.
- Components included in the ActionBar are:

1. **App Icon:** Display the branding logo/icon of the application.
2. **View Controls:** Section that displays the name of the application or current activity. Developers can also include spinner or tabbed navigation for switching between views.
3. **Action Button:** Contains some important actions/elements of the app that may be required to the users frequently.
4. **Action Overflow:** Include other actions that will be displayed as a menu. Move less often used actions to the action overflow.



Changing the ActionBar Title

- The ActionBar title displayed at the top of the screen is governed by the **AndroidManifest.xml** file within the activity nodes.
- In the example below, the activity "FirstActivity" will have an ActionBar with the string value of the resource identified by @string/activity_name.
- If the value of that resource is "Foo," the string displayed in the ActionBar for this activity will be "Foo."

```
<application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name="com.codepath.example.simpleapp.FirstActivity"
            android:label="@string/activity_name" >
        </activity>
</application>
```

- Change the **android:label or android:icon** to modify the ActionBar title or icon for a given activity or for the application as a whole.
- In any Java activity, you can also call **getSupportActionBar()** to retrieve a reference to the ActionBar and modify or access any properties of the ActionBar at runtime:

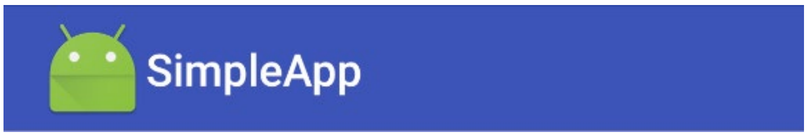
```
ActionBar actionBar = getSupportActionBar(); // or getActionBar();
getSupportActionBar().setTitle("My new title"); // set the top title
String title = actionBar.getTitle().toString(); // get the title
actionBar.hide(); // or even hide the actionbar
```

Displaying ActionBar Icon

The icon can be added with:

```
getSupportActionBar().setDisplayShowHomeEnabled(true);
getSupportActionBar().setLogo(R.mipmap.ic_launcher);
getSupportActionBar().setDisplayUseLogoEnabled(true);
```

The above code results in:



Adding Action Items

When you want to add primary actions to the ActionBar, you add the items to the activity context menu and if properly specified, they will automatically appear at the top right as icons in the ActionBar.

An activity populates the ActionBar from within the onCreateOptionsMenu() method:

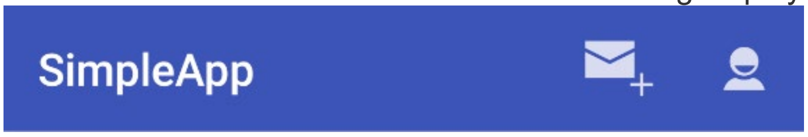
```
public class MainActivity extends AppCompatActivity {
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
}
```

Entries in the action bar are typically called actions. Use this method to inflate a menu resource that defines all the action items within a res/menu/menu_main.xml file, for example:

```
<!-- Menu file for `activity_movies.xml` is located in a file
     such as `res/menu/menu_movies.xml`  -->
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/miCompose"
        android:icon="@drawable/ic_compose"
        app:showAsAction="ifRoom"
        android:title="Compose">
    </item>
    <item
        android:id="@+id/miProfile"
        android:icon="@drawable/ic_profile"
        app:showAsAction="ifRoom|withText"
        android:title="Profile">
    </item>
</menu>
```

You also should note that the xmlns:app namespace must be defined in order to leverage the showAsAction option. The reason is that a compatibility library is used to support the showAsAction="ifRoom" option. This option is needed to show the item directly in the action bar as an icon. If there's not enough room for the item in the action bar, it will appear in the action overflow. If withText is specified as well (as in the second item), the text will be displayed with the icon.

The above code results in two action icons being displayed:



Creating the user interface programmatically

- All the UIs are created using XML.
- Besides using XML you can also create the UI using code.
- This approach is useful if your UI needs to be dynamically generated during runtime.
- For example, suppose you are building a cinema ticket reservation system and your application displays the seats of each cinema using buttons. In this case, you need to dynamically generate the UI based on the cinema selected by the user.
- The following Try It Out demonstrates the code needed to dynamically build the UI in your activity.
- To create a user interface programmatically in Android Studio, you typically follow these steps:

  1. *Create a Layout Container*: You'll need a layout container to hold your UI elements programmatically. This can be a LinearLayout, RelativeLayout, ConstraintLayout, etc. You can create it directly in your activity's XML layout file or programmatically in your Java/Kotlin code.

  2. *Instantiate UI Elements*: Create instances of the UI elements you want to include in your layout. For example, TextView, Button, EditText, etc.

  3. *Set Layout Parameters*: For each UI element, create layout parameters specifying how it should be positioned and sized within the layout container.

  4. *Add UI Elements to Layout*: Add the UI elements to the layout container using the addView() method.

5. ***Set Content View*:** If you created the layout container programmatically, set it as the content view of your activity using the setContentView() method.

Creating the UI via Code (UICode.zip)

1. Using Android Studio, create a new Android project and name it **UICode**.

2. In the **MainActivity.java** file, add the bold statements in the following code:

```
import android.support.v7.app.AppCompatActivity; import android.os.Bundle;

import android.support.v7.widget.LinearLayoutCompat;

import android.widget.Button;

import android.widget.LinearLayout;

import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) { super.
OnCreate(savedInstanceState);
LinearLayoutCompat.LayoutParams params = new
LinearLayoutCompat.LayoutParams(
LinearLayoutCompat.LayoutParams.WRAP_CONTENT,
LinearLayoutCompat.LayoutParams.WRAP_CONTENT);

//---create a layout---
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
//---create a text view---

        TextView tv = new TextView(this);
        tv.setText("This is a TextView");
        tv.setLayoutParams(params);

//---create a button---

        Button btn = new Button(this);
        btn.setText("This is a Button");
        btn.setLayoutParams(params);

//---adds the textview---

         layout.addView(tv);

//---adds the button--- layout.addView(btn);
//---create a layout param for the layout---

        LinearLayoutCompat.LayoutParams layoutParam = new
        LinearLayoutCompat.LayoutParams(
        LinearLayoutCompat.LayoutParams.WRAP_CONTENT,
        LinearLayoutCompat.LayoutParams.WRAP_CONTENT );
        this.addContentView(layout, layoutParam);
        } }
```

3. Press Shift+F9 to debug the application on the Android emulator.

How It Works

In this example, you first commented out the setContentView() statement so that it does not load the UI from the activity_main.xml file.
You then created a LayoutParams object to specify the layout parameter that can be used by other views (which you will create next):

```
//---create a layout param for the layout---
        LinearLayoutCompat.LayoutParams layoutParam = new
        LinearLayoutCompat.LayoutParams(
        LinearLayoutCompat.LayoutParams.WRAP_CONTENT,
        LinearLayoutCompat.LayoutParams.WRAP_CONTENT );
```

You also created a LinearLayout object to contain all the views in your activity:

```
//---create a layout---
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
```

Next, you created a TextView and a Button view:

```
//---create a textview---
        TextView tv = new TextView(this);
         tv.setText("This is a TextView"); tv.setLayoutParams(params);
 //---create a button---

        Button btn = new Button(this); btn.setText("This is a Button");
        btn.setLayoutParams(params);
```

You then added them to the LinearLayout object:

```
//---adds the textview---
        layout.addView(tv);
 //---adds the button---
         layout.addView(btn);
```

You also created a LayoutParams object to be used by the LinearLayout object:

```
//---create a layout param for the layout---
        LinearLayout.LayoutParams layoutParam =
        new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT,
        LayoutParams.WRAP_CONTENT );
```

Finally, you added the LinearLayout object to the activity:

```
       this.addContentView(layout, layoutParam);
```

As you can see, using code to create the UI is quite a laborious affair. Hence, you should dynamically generate your UI using code only when necessary.

Listening for UI notifications

INTRODUCTION

There are two levels of Android user interface with which users interact and they are as follows:

1. Activity level
2. View level

ACTIVITY LEVEL

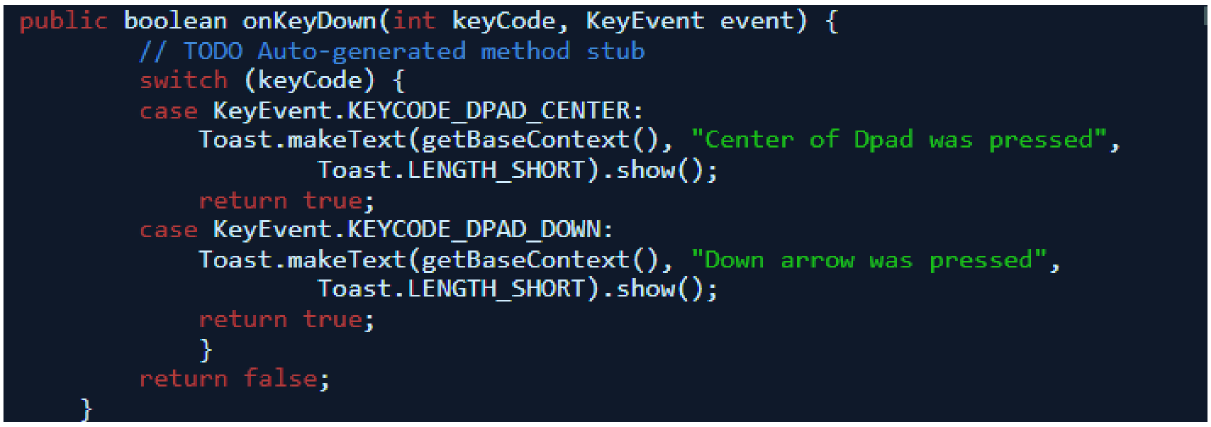
At activity level, there are certain methods in Activity class which we can override. Some of the genuine methods are as follow:

It's often used to detect when a user releases a key on a hardware keyboard or a software keyboard on the screen. This method is useful for handling input from keyboard events, such as responding to specific keys being pressed or released.

- **onKeyUp():** This is called when a key was released. This is not handled by any of the views inside the activity.
- **onKeyDown()**: This is called when a key was pressed. This is not handled by any of the views inside the activity.

- **onMenuItemSelected()**: This is called when any item of the menu panel is pressed by user.
- **onMenuOpened():** This method is called when user opens the panel's menu.

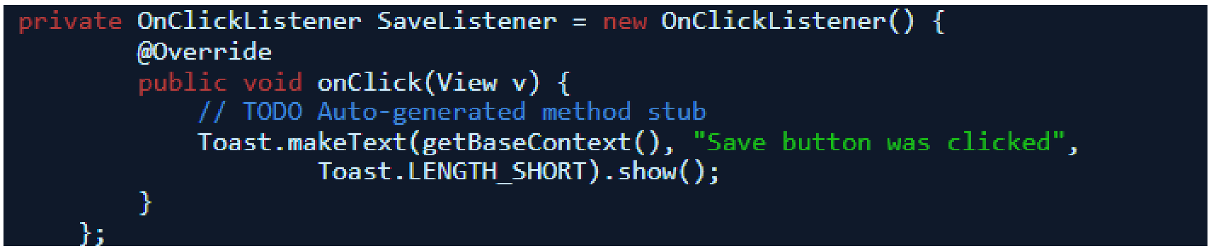
The following code snippet is an instance of such an implementation:



```java
public boolean onKeyDown(int keyCode, KeyEvent event) {
        // TODO Auto-generated method stub
        switch (keyCode) {
        case KeyEvent.KEYCODE_DPAD_CENTER:
            Toast.makeText(getBaseContext(), "Center of Dpad was pressed",
                    Toast.LENGTH_SHORT).show();
            return true;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            Toast.makeText(getBaseContext(), "Down arrow was pressed",
                    Toast.LENGTH_SHORT).show();
            return true;
            }
        return false;
    }
```

VIEW LEVEL

- ➢ When any user interacts with a view, the corresponding view fires event.
- ➢ When a user touches a button or an image button or any such view we have to service the related service so that appropriate action can be performed.
- ➢ For this, events need to be registered. For a button we will have code like this:



```java
private OnClickListener SaveListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            Toast.makeText(getBaseContext(), "Save button was clicked",
                    Toast.LENGTH_SHORT).show();
        }
    };
```

<u>// getBaseContext() – </u>this method is used to retrieve the base context of an activity or application. This context provides access to application-specific resources and operations such **as accessing files, launching activities or accessing system services.**

<u>Designing Your User Interface with Views</u>

View is the basic building block of UI(User Interface) in android. View refers to the **android.view.View class**, which is the super class for all the GUI components like TextView, ImageView, Button etc.

➤➤**Basic views**—Commonly used views, such as the TextView, EditText, and Button views.

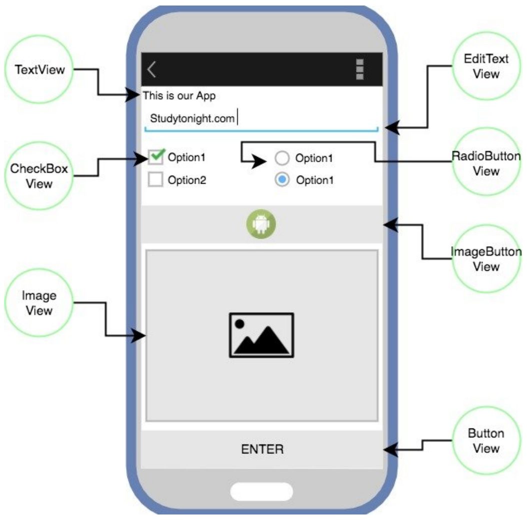
➤➤ **Picker views**—Views that enable users to select from a list, such as the TimePicker and DatePicker views.

➤➤ **List views**—Views that display a long list of items, such as the ListView and the SpinnerView views.

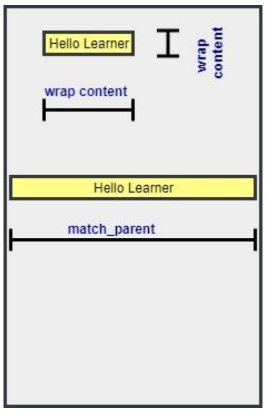
➤➤ **Specialized fragments**—Special fragments that perform specific functions.

Using Basic Views

```
TextView
EditText
Button
ImageButton
CheckBox
ToggleButton
RadioButton
RadioGroup
```



- These basic views enable you to display text information, as well as perform some basic selection.
- View class extends Object class and implements **Drawable.Callback, KeyEvent.Callback and AccessibilityEventSource.**
- match_parent means it will occupy the complete space available on the display of the device. Whereas, wrap_content means it will occupy only that much space as required for its content to display.



TextView View

When you create a new Android project, Android Studio always creates the activity_main.xml file(located in the res/layout folder), which contains a <TextView> element:

```
        <?xml version="1.0" encoding="utf-8"?>
        <LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
        <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
        </LinearLayout>
```

You use the TextView view to display text to the user. This is the most basic view and one that you will frequently use when you develop Android applications. If you need to allow users to edit the text displayed, you should use the subclass of ***TextView—EditText.***

Button, imageButton, edittext, CheckBox, toggleButton, radioButton, and radiogroup Views

Besides the TextView view, which you will likely use the most often, there are some other basic views that you will find yourself frequently using:

- ➤➤ Button—Represents a push-button widget.
- ➤➤ ImageButton—Similar to the Button view, except that it also displays an image.
- ➤➤ EditText—A subclass of the TextView view, which allows users to edit its text content.
- ➤➤ CheckBox—A special type of button that has two states: checked or unchecked.
- ➤➤ RadioGroup and RadioButton—The RadioButton has two states: either checked or unchecked. A RadioGroup is used to group one or more RadioButton views, thereby allowing only one RadioButton to be checked within the RadioGroup.
- ➤➤ ToggleButton—Displays checked/unchecked states using a light indicator.

Using the Basic Views (BasicViews1.zip)

1. Using Android Studio, create an Android project and name it **BasicViews1**.
2. Modify the activity_main.xml file located in the res/layout folder by adding the following elements shown in bold:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button android:id="@+id/btnSave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="save" />
<Button android:id="@+id/btnOpen"
    android:layout_width="wrap_content"


    android:layout_height="wrap_content"
    android:text="Open" />
<ImageButton android:id="@+id/btnImg1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:src="@mipmap/ic_launcher" />
<EditText android:id="@+id/txtName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
<CheckBox android:id="@+id/chkAutosave"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Autosave" />
<CheckBox android:id="@+id/star"
    style="?android:attr/starStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<RadioGroup android:id="@+id/rdbGp1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
```

3. To see the views in action, debug the project in Android Studio by pressing Shift+F9. shows the various views displayed in the Android emulator.
4. Click each of the views and note how they vary in look and feel. Figure 5-2 shows the following
changes to the view:

➤➤ The first CheckBox view (Autosave) is checked.

➤➤ The second CheckBox View (star) is selected.

➤➤ The second RadioButton (Option 2) is selected.

➤➤ The ToggleButton is turned on.

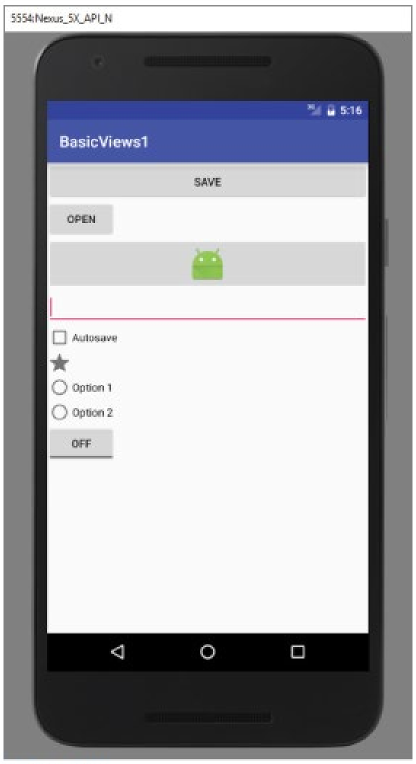


FIGURE 5-1

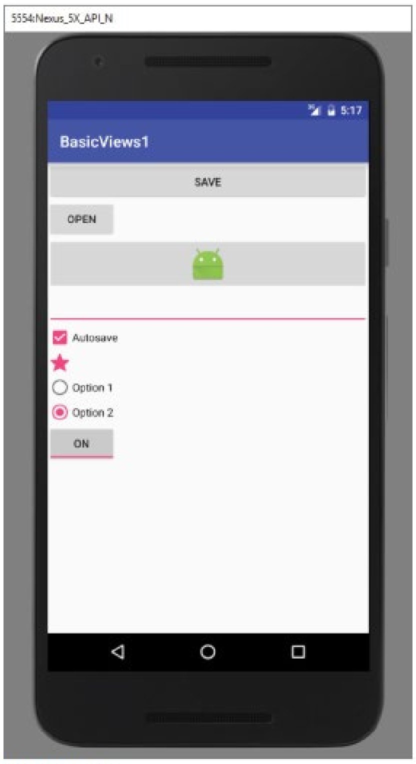


FIGURE 5-2

- set the image through the *src* attribute.
- layout_height has been set to *wrap_content* so that the text entry location automatically adjusts to fit the amount of text entered by the user.
- If you do not like the default look of the CheckBox, you can apply a style attribute so that the check mark is replaced by another image, such as a star

  style="?android:attr/starStyle

<u>Using picker Views</u>

Selecting a date and time is one of the common tasks you need to perform in a mobile application.
Android supports this functionality through the TimePicker and DatePicker views.

Timepicker View

- The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode. The following Try It Out shows you how to use the TimePicker in the latest version of the Android SDK.
- When you are creating the project for this sample, be sure that you choose an SDK that is level 23 or greater.
- The ***android.widget.DatePicker*** is the subclass of **FrameLayout** class.

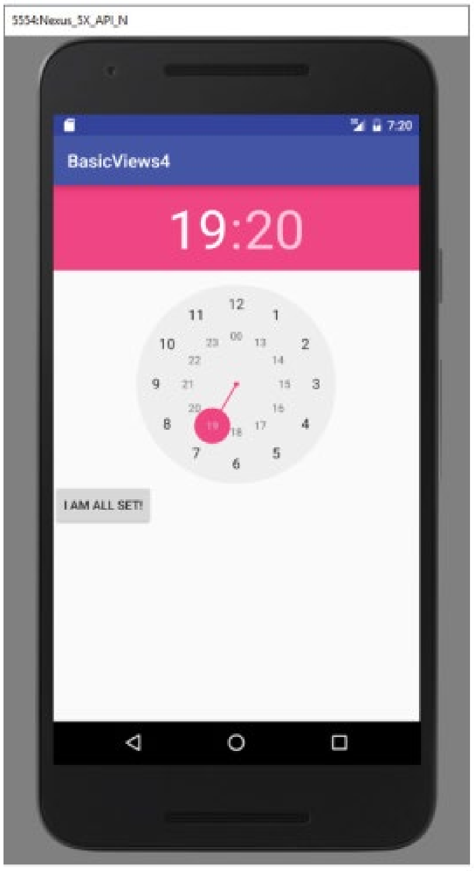
Using the timepicker View

1. Using Android Studio, create an Android project and name it **BasicViews4**.

2. Modify the activity_main.xml file located in the res/layout folder by adding the following bolded lines:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
<TimePicker android:id="@+id/timePicker"
    android:layout_width="wrap_content"

      android:layout_height="wrap_content" />
  <Button android:id="@+id/btnSet"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am all set!"
      android:onClick="onClick" />
  </LinearLayout>
```

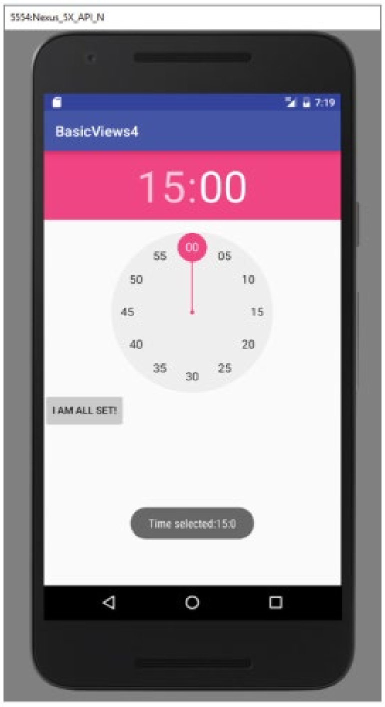
3. Press Shift+F9 to debug the application on the Android emulator. Figure 5-8 shows the TimePicker in action. You can use the numeric keypad or the time widget on the screen to change the hour and minute.



4. Back in Android Studio, add the following bolded statements to the **MainActivity.java file:**

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TimePicker;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    TimePicker timePicker;


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        timePicker = (TimePicker) findViewById(R.id.timePicker);
        timePicker.setIs24HourView(true);
    }

    public void onClick(View view) {
        Toast.makeText(getBaseContext(),
                "Time selected:" +
                timePicker.getHour() +
        ":" + timePicker.getMinute(),
                Toast.LENGTH_SHORT).show();
    }

}
```

5. Press Shift+F9 to debug the application on the Android emulator. This time, the *TimePicker* is displayed in the 24-hour format. Clicking the Button displays the time that you have set in the *TimePicker*



How It Works

- The TimePicker displays a standard UI to enable users to set a time. By default, it displays the time in the AM/PM format.
- If you want to display the time in the 24-hour format, you can use the setIs24HourView() method.

Datepicker View

Another view that is similar to the *TimePicker* is the *DatePicker*. Using the *DatePicker*, you can enable users to select a particular date on the activity. The following Try It Out shows you how to use the *DatePicker*.

Using the Datepicker View

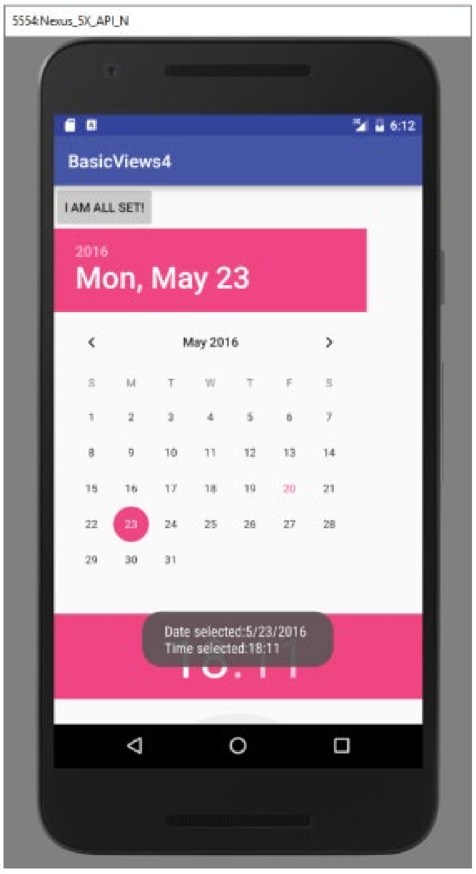
1. Using the BasicViews4 project created earlier, modify the activity_main.xml file as shown here:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
<Button android:id="@+id/btnSet"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am all set!"
    android:onClick="onClick" />
<DatePicker android:id="@+id/datePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<TimePicker android:id="@+id/timePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

2. Add the following bolded statements to the MainActivity.java file:

```
import android.app.TimePickerDialog;
import android.icu.text.SimpleDateFormat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TimePicker;
import android.widget.Toast;

import java.util.Date;

public class MainActivity extends AppCompatActivity {
    TimePicker timePicker;
    DatePicker datePicker;

    int hour, minute;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        timePicker = (TimePicker) findViewById(R.id.timePicker);
        timePicker.setIs24HourView(true);

        datePicker = (DatePicker) findViewById(R.id.datePicker);
    }
    private TimePickerDialog.OnTimeSetListener mTimeSetListener =
            new TimePickerDialog.OnTimeSetListener()
            {
                public void onTimeSet(
                        TimePicker view, int hourOfDay, int minuteOfHour)
                {
                    hour = hourOfDay;
                    minute = minuteOfHour;

                    SimpleDateFormat timeFormat = new SimpleDateFormat("hh:mm aa");
                    Date date = new Date();
                    String strDate = timeFormat.format(date);

                    Toast.makeText(getBaseContext(),
                            "You have selected " + strDate,
                            Toast.LENGTH_SHORT).show();
                }
            };
    public void onClick(View view) {
        Toast.makeText(getBaseContext(),
                "Date selected:" + (datePicker.getMonth() + 1) +
                        "/" + datePicker.getDayOfMonth() +
                        "/" + datePicker.getYear() + "\n" +
                        "Time selected:" + timePicker.getHour() +
                        ":" + timePicker.getMinute(),
                Toast.LENGTH_SHORT).show();
    }
}
```

3. Press Shift+F9 to debug the application on the Android emulator. After the date is set, clicking the Button displays the date set.



Date selected:5/23/2016
Time selected:18:11

How It Works

As with the `TimePicker`, you call the `getMonth()`, `getDayOfMonth()`, and `getYear()` methods to get the month, day, and year, respectively:

```
"Date selected:" + (datePicker.getMonth() + 1) +
 "/" + datePicker.getDayOfMonth() + "/" +
datePicker.getYear() +
 "\n" +
```

Note that the `getMonth()` method returns 0 for January, 1 for February, and so on. This means you need to increment the result of this method by one to get the corresponding month number.
Like the `TimePicker`, you can also display the `DatePicker` in a dialog window.

Using list Views to Display long lists

- List of scrollable items can be displayed in Android using **ListView.**
- It helps you to displaying the data in the form of a scrollable list.
- Users can then select any list item by clicking on it.
- ListView is default scrollable so we do not need to use scroll View or anything else with ListView. ListView is widely used in android applications.
- In Android, there are two types of list views:
  ListView and SpinnerView.
- A very common example of ListView is your phone contact book, where you have a list of your contacts displayed in a ListView and if you click on it then user information is displayed.

- A ListView is a type of **<u>AdapterView</u>.**
- In Android, ***AdapterView is an abstract class*** that acts as a parent for several view classes like **ListView, GridView, and Spinner**. It's used to display data in a scrollable format by connecting data to the respective view.
- Using adapter, items are inserted into the list from an array or database. For displaying the items in the list method ***setAdaptor()*** is used.
- The main purpose of the adapter is to fetch data from an array or database and insert each item that placed into the list for the desired result.
- The ***ListView*** and ***GridView*** are subclasses of ***AdapterView***
- Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adapters are **<u>ArrayAdapter,Base Adapter, CursorAdapter, SimpleCursorAdapter,SpinnerAdapter</u>** and ***WrapperListAdapter***. We will see separate examples for both the adapters.
- To display a list, you can include a list view in your layout XML file:

```
<ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

ListView Attributes

Following are the important attributes specific to GridView −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:divider** This is drawable or color to draw between list items.
3	**android:dividerHeight** This specifies height of the divider. This could be in px, dp, sp, in, or mm.
4	**android:entries** Specifies the reference to an array resource that will populate the ListView.
5	**android:footerDividersEnabled** When set to false, the ListView will not draw the divider before each footer view. The default value is true.

6	**android:headerDividersEnabled** When set to false, the ListView will not draw the divider after each header view. The default value is true.

In android commonly used adapters are:

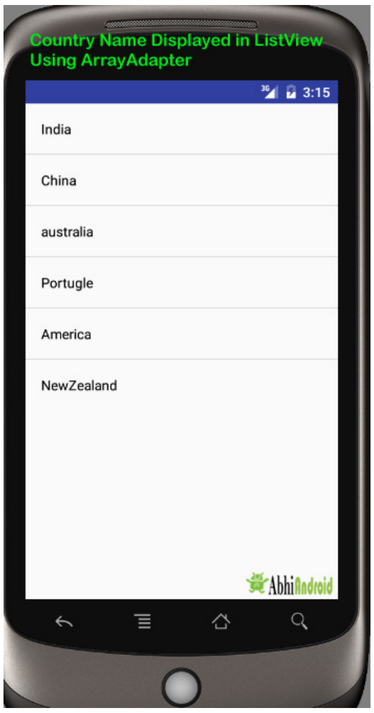
1. Array Adapter
2. Base Adapter

1.Array Adapter:

Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter. For instance, list of phone contacts, countries or names.

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,R
.id.textView,StringArray);
```

Example of list view using Array Adapter:



2.Base Adapter:

Whenever you need a customized list you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying a custom list item.

```
public class CustomAdapter extends BaseAdapter {
@Override
```

```
public int getCount() {
return 0;
}
@Override
public Object getItem(int i) {
return null;
}
@Override
public long getItemId(int i) {
return 0;
}
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
return null;
}
```

1. getCount():

The getCount() function returns the total number of items to be displayed in a list.

```
public int getCount() {
int count=arrayList.size(); //counts the total number of elements from the arrayList
return count;//returns the total count to adapter
}
```

2. getView(int i, View view, ViewGroup viewGroup):

This function is automatically called when the list item view is ready to be displayed or about to be displayed.

```
public View getView(int i, View view, ViewGroup viewGroup) {
view = inflter.inflate(R.layout.activity_gridview, null);//set layout for displaying i
tems
ImageView icon = (ImageView) view.findViewById(R.id.icon);//get id for image view
icon.setImageResource(flags[i]);//set image of the item’s
return view;
}
```

3. getItem(int i):

This function is used to Get the data item associated with the specified position in the data set to obtain the corresponding data of the specific location in the collection of data items.

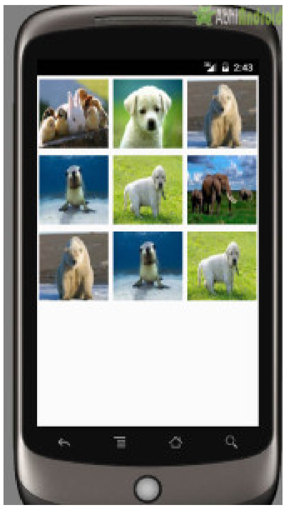
```
@Override
public Object getItem(int i) {
return arrayList.get(i);
}
```

4. getItemId(int i):

As for the getItemId (int position), it returns the corresponding to the position item ID. The function returns a long value of item position to the adapter.

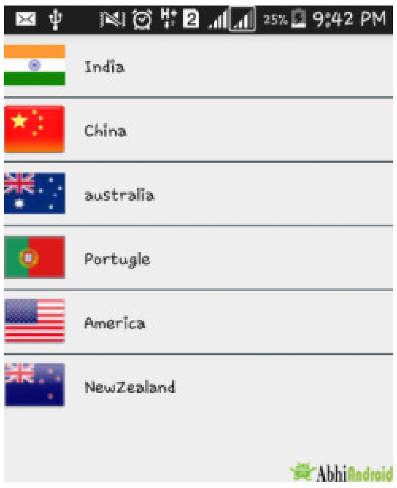
```
@Override
public long getItemId(int i) {
return i;
}
```

Below is the final output and code with explanation:



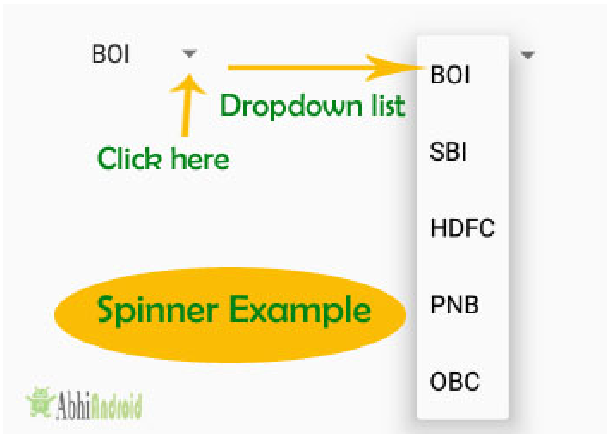
Example of list view using Custom adapter(Base adapter):

In this example we display a list of countries with flags. For this, we have to use custom adapter.



Spinner View

In Android, Spinner provides a quick way to select one value from a set of values. Android spinners are nothing but the drop down-list. In a default state, a spinner shows its currently selected value. It provides a easy way to select a value from a list of values.



Here is the XML basic code for Spinner:

```
<Spinner
android:id="@+id/simpleSpinner "
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
```

Important Note: To fill the data in a spinner we need to implement an adapter class. A spinner is mainly used to display only text field so we can implement Array Adapter for that. We can also use Base Adapter and other custom adapters to display a spinner with more customize list. Suppose if we need to display a textview and a imageview in spinner item list then array adapter is not enough for that. Here we have to implement custom adapter in our class. Below image of Spinner and Custom Spinner will make it more clear.

Understanding specialized Fragments

As you have learned, fragments are really "mini-activities" that have their own life cycles. To create a fragment, you need a class that extends the Fragment base class. In addition to the Fragment base class, you can also extend from some other subclasses of the Fragment base class to create more specialized fragments. The following sections discuss the three subclasses of Fragment:

- ***ListFragment***
- ***DialogFragment***
- ***PreferenceFragment***

Using a ListFragment

- A list fragment is a fragment that contains a ListView, which displays a list of items from a data source, such as an array or a Cursor.
- A list fragment is useful because it's common to have one fragment that contains a list of items, and another fragment that displays details about the selected posting.
- In the chat application, a ListFragment could be used to display a list of messages exchanged between users in a chat conversation. Each item in the list represents a message, showing the sender's name, message content, timestamp, and possibly other metadata such as profile pictures.
- To create a list fragment, you need to extend the *ListFragment* base class.

Creating and Using a List Fragment

1. Using Android Studio, create an Android project and name it **ListFragmentExample**.

2. Modify the ***activity_main.xml*** file as shown in bold.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
    <fragment
        android:name="com.jfdimarzio.listfragmentexample.Fragment1"
        android:id="@+id/fragment1"
        android:layout_weight="0.5"
        android:layout_width="0dp"
        android:layout_height="200dp" />
    <fragment
        android:name="com.jfdimarzio.listfragmentexample.Fragment1"
        android:id="@+id/fragment2"
        android:layout_weight="0.5"
        android:layout_width="0dp"
        android:layout_height="300dp" />
</LinearLayout>
```

3. Add an XML file to the ***res/layout*** folder and name it **fragment1.xml**.

4. Populate the ***fragment1.xml*** as follows:
<?xml version="1.0" encoding="utf-8"?>

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:drawSelectorOnTop="false"/>
</LinearLayout>
```

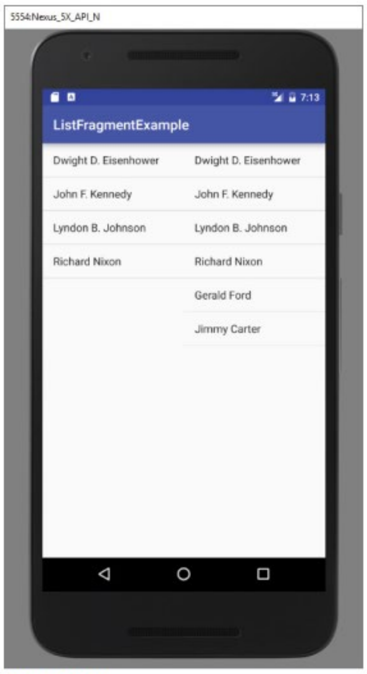
5. Add a Java Class file to the package and name it **Fragment1**.

6. Populate the ***Fragment1.java*** file as follows:

```
import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
public class Fragment1 extends ListFragment {
    String[] presidents = {
        "Dwight D. Eisenhower",
        "John F. Kennedy",
        "Lyndon B. Johnson",
        "Richard Nixon",
        "Gerald Ford",
        "Jimmy Carter",
        "Ronald Reagan",
        "George H. W. Bush",
        "Bill Clinton",
        "George W. Bush",
        "Barack Obama"
    };
    @Override
    public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment1, container, false);
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<String>(getActivity(),
            android.R.layout.simple_list_item_1, presidents));
    }

    public void onListItemClick(ListView parent, View v,
    int position, long id)
    {
        Toast.makeText(getActivity(),
            "You have selected " + presidents[position],
            Toast.LENGTH_SHORT).show();
    }
```

7. Press Shift+F9 to debug the application on the Android emulator. Figure 5-18 shows the two list fragments displaying the two lists of presidents' names.

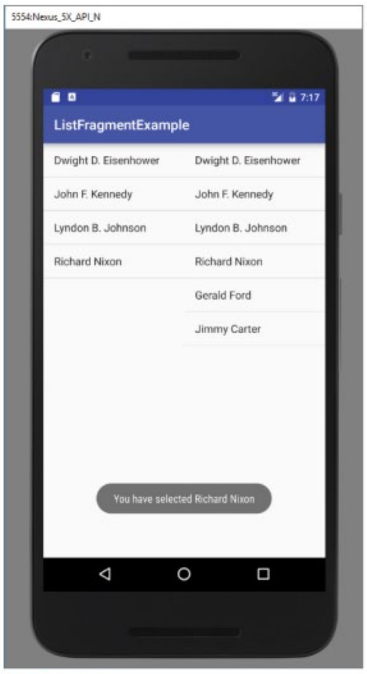
8. Click any of the items in the two ListView views, and you see a message


5554:Nexus_5X_API_N

ListFragmentExample

Dwight D. Eisenhower	Dwight D. Eisenhower
John F. Kennedy	John F. Kennedy
Lyndon B. Johnson	Lyndon B. Johnson
Richard Nixon	Richard Nixon
	Gerald Ford
	Jimmy Carter

FIGURE 5-18


5554:Nexus_5X_API_N

ListFragmentExample

Dwight D. Eisenhower	Dwight D. Eisenhower
John F. Kennedy	John F. Kennedy
Lyndon B. Johnson	Lyndon B. Johnson
Richard Nixon	Richard Nixon
	Gerald Ford
	Jimmy Carter

You have selected Richard Nixon

FIGURE 5-19

Using a DialogFragment

- A dialog fragment floats on top of an activity and is displayed modally.
- Dialog fragments are useful when you need to obtain the user's response before continuing with execution.
- To create a dialog fragment, you must extend the ***DialogFragment*** base class.

Creating and Using a Dialog Fragment

1. Using Android Studio, create an Android project and name it **DialogFragmentExample**.

2. Add a Java Class file under the package and name it **Fragment1**.

3. Populate the Fragment1.java file as follows:

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;

public class Fragment1 extends DialogFragment {
    static Fragment1 newInstance(String title) {
        Fragment1 fragment = new Fragment1();
        Bundle args = new Bundle();
        args.putString("title", title);
        fragment.setArguments(args);
        return fragment;
    }
```

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    String title = getArguments().getString("title");
    return new AlertDialog.Builder(getActivity())
            .setIcon(R.mipmap.ic_launcher)
            .setTitle(title)
            .setPositiveButton("OK",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
                                            int whichButton) {
                            ((MainActivity)
                                    getActivity()).doPositiveClick();
```

4. Populate the MainActivity.java file as shown here in bold:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Fragment1 dialogFragment = Fragment1.newInstance(
                "Are you sure you want to do this?");
         dialogFragment.show(getFragmentManager(), "dialog");
    }

    public void doPositiveClick() {
        //---perform steps when user clicks on OK---
        Log.d("DialogFragmentExample", "User clicks on OK");
    }
    public void doNegativeClick() {
        //---perform steps when user clicks on Cancel---
        Log.d("DialogFragmentExample", "User clicks on Cancel");
    }
}
```

5. Press Shift+F9 to debug the application on the Android emulator. Figure 5-20 shows the fragment displayed as an alert dialog. Click either OK or Cancel and observe the message displayed.

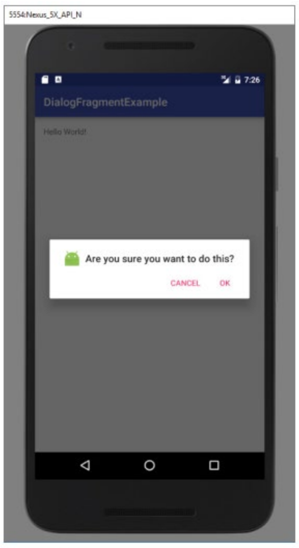


FIGURE 5-20

Using a preferenceFragment

- In your Android applications you provide preferences for users to personalize the application.
- For example, you might allow users to save the login credentials that they use to access their web resources.
- Also, you could save information, such as how often the feeds must be refreshed and so on.
- In the social media app, a PreferenceFragment could be used to display the app's settings screen, These XML files define preferences such as notification settings, privacy settings, theme preferences, etc.
- In Android, you can use the **PreferenceActivity** base class to display an activity for the user to edit the preferences. In Android 3.0 and later, you can use the **PreferenceFragment** class to do the same thing.
- In preferences there are different types of preferences which are listed below :
    - *EditTextPreference:* this is used to get the text from the user.
    - *ListPreference:* this option is used to display a dialog with the list of options to choose from.
    - *CheckBoxPreference:* this option is used to display a checkbox to toggle a setting.
    - *SwitchPreference:* this option is used to turn the switch on and off.
    - *RingtonePreference:* this option is used to open the ringtone page of your device.
    - Preference with an Intent action android.intent.action.VIEW – to open an external browser navigating to an URL.

Creating and Using a preference Fragment

1. Using Android Studio, create an Android project and name it **PreferenceFragmentExample**.
2. Create a new xml directory under the res folder and then add a new XML resource file to it.
Name the XML file **preferences.xml**.
3. Populate the preferences.xml file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Category 1">
        <CheckBoxPreference
            android:title="Checkbox"
            android:defaultValue="false"
            android:summary="True of False"
            android:key="checkboxPref" />
        </PreferenceCategory>

    <PreferenceCategory android:title="Category 2">
        <EditTextPreference
            android:name="EditText"
            android:summary="Enter a string"
            android:defaultValue="[Enter a string here]"
            android:title="Edit Text"
            android:key="editTextPref" />

        <RingtonePreference
            android:name="Ringtone Preference"
            android:summary="Select a ringtone"
            android:title="Ringtones"
            android:key="ringtonePref" />
        <PreferenceScreen
            android:title="Second Preference Screen"
            android:summary=
                "Click here to go to the second Preference Screen"
            android:key="secondPrefScreenPref">
            <EditTextPreference
                android:name="EditText"
                android:summary="Enter a string"
                android:title="Edit Text (second Screen)"
                android:key="secondEditTextPref" />
        </PreferenceScreen>
    </PreferenceCategory>

</PreferenceScreen>
```

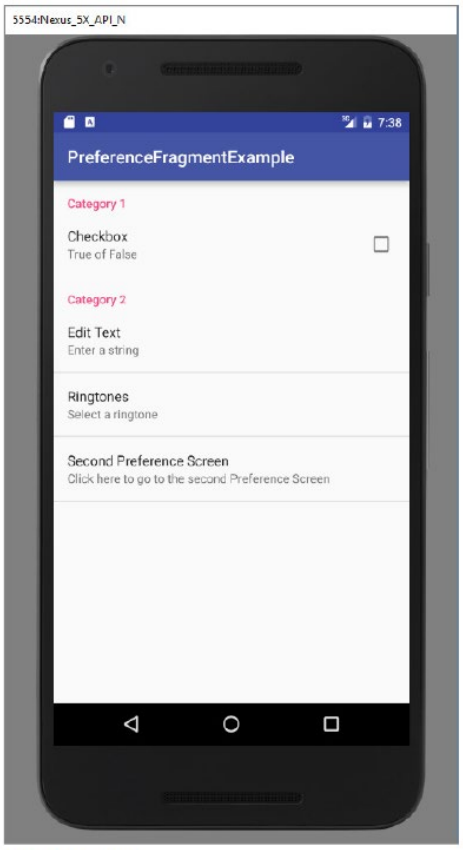
4. Add a Java Class file to the package and name it Fragment1.
5. Populate the Fragment1.java file as follows:

```
import android.os.Bundle;
import android.preference.PreferenceFragment;
public class Fragment1 extends PreferenceFragment {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//---load the preferences from an XML file---
addPreferencesFromResource(R.xml.preferences);
        }
}
```

6. Modify the `MainActivity.java` file as shown in bold:

```
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        FragmentManager fragmentManager = getFragmentManager();
         FragmentTransaction fragmentTransaction =
                fragmentManager.beginTransaction();
         Fragment1 fragment1 = new Fragment1();
         fragmentTransaction.replace(android.R.id.content, fragment1);
         fragmentTransaction.addToBackStack(null);
         fragmentTransaction.commit();
    }
}
```

7. Press Shift+F9 to debug the application on the Android emulator. Figure 5-21 shows the preference fragment displaying the list of preferences that the user can modify.
8. When the Edit Text preference is clicked, a pop-up is displayed (see Figure 5-22).
9. Clicking Edit Text (Second Screen) causes a second preference screen to be displayed (see Figure 5-23).
10. To dismiss the preference fragment, click the Back button on the emulator.


5554:Nexus_5X_API_N

7:38

PreferenceFragmentExample

Category 1

Checkbox
True or False

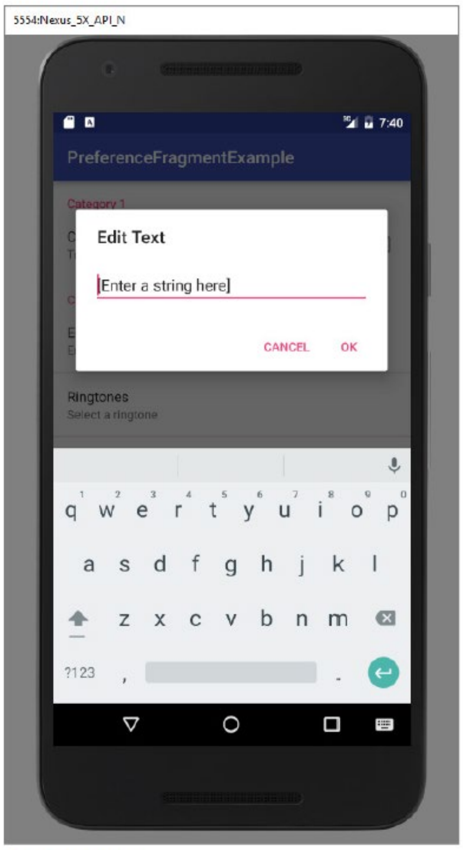
Category 2

Edit Text
Enter a string

Ringtones
Select a ringtone

Second Preference Screen
Click here to go to the second Preference Screen

FIGURE 5-21


5554:Nexus_5X_API_N

7:40

PreferenceFragmentExample

Edit Text

Enter a string here

CANCEL OK

Ringtones
Select a ringtone

FIGURE 5-22

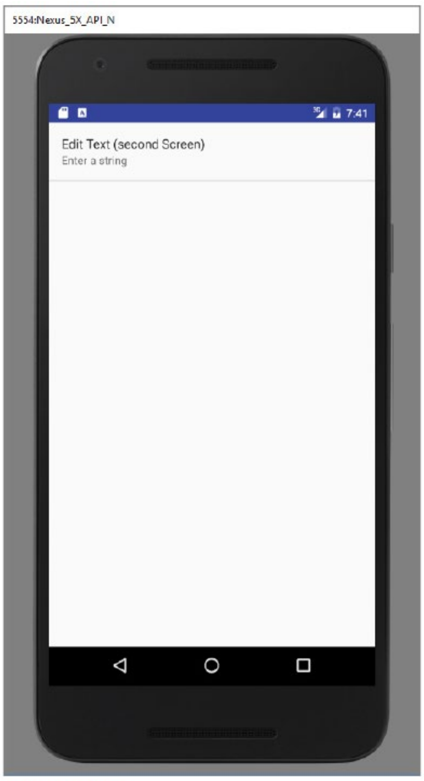
How It Works

To create a list of preferences in your Android application, you first need to create the preferences .xml file and populate it with the various XML elements. This XML file defines the various items that you want to persist in your application.
To create the preference fragment, you must extend the PreferenceFragment base class:

```
public class Fragment1 extends PreferenceFragment {
}
```

To load the preferences file in the preference fragment, use the addPreferencesFromResource() method:

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//---load the preferences from an XML file---
addPreferencesFromResource(R.xml.preferences);
}
```


5554:Nexus_5X_API_N

7:41

Edit Text (second Screen)

Enter a string

FIGURE 5-23

To display the preference fragment in your activity, you can make use of the FragmentManager and the FragmentTransaction classes:

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
Fragment1 fragment1 = new Fragment1();
fragmentTransaction.replace(android.R.id.content, fragment1);
fragmentTransaction.addToBackStack(null);
fragmentTransaction.commit();
```

You need to add the preference fragment to the back stack using the addToBackStack() method so that 11the user can dismiss the fragment by clicking the Back button.

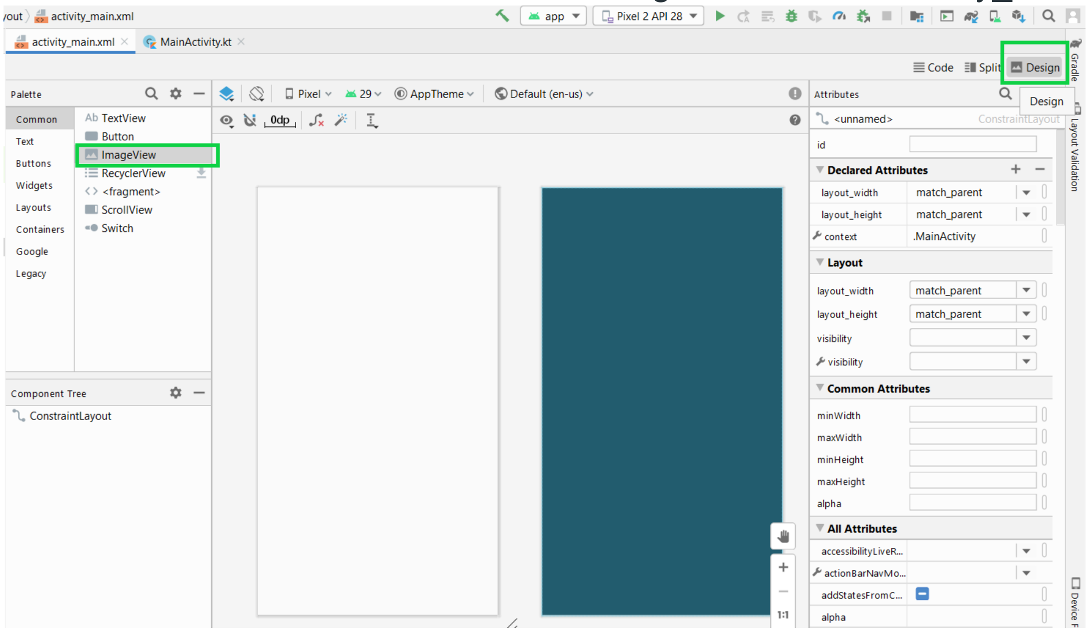
ImageView in Android

- **ImageView** class is used to display any kind of image resource in the android application either it can be **android.graphics.Bitmap** or **android.graphics.drawable.Drawable** (it is a general abstraction for anything that can be drawn in Android).
- ImageView class or **android.widget.ImageView** inherits the **android.view.View** class which is the subclass of Java.
- Moreover, ImageView is also used to control the size and movement of an image.

Adding an ImageView to an Activity

- Whenever ImageView is added to an activity, it means there is a requirement for an image resource.
- Thus it is oblivious to provide an Image file to that ImageView class.
- It can be done by adding an image file that is present in the Android Studio itself or we can add our own image file.
- Android Studio owns a wide range of drawable resources which are very common in the android application layout.
- The following are the steps to add a drawable resource to the ImageView class.

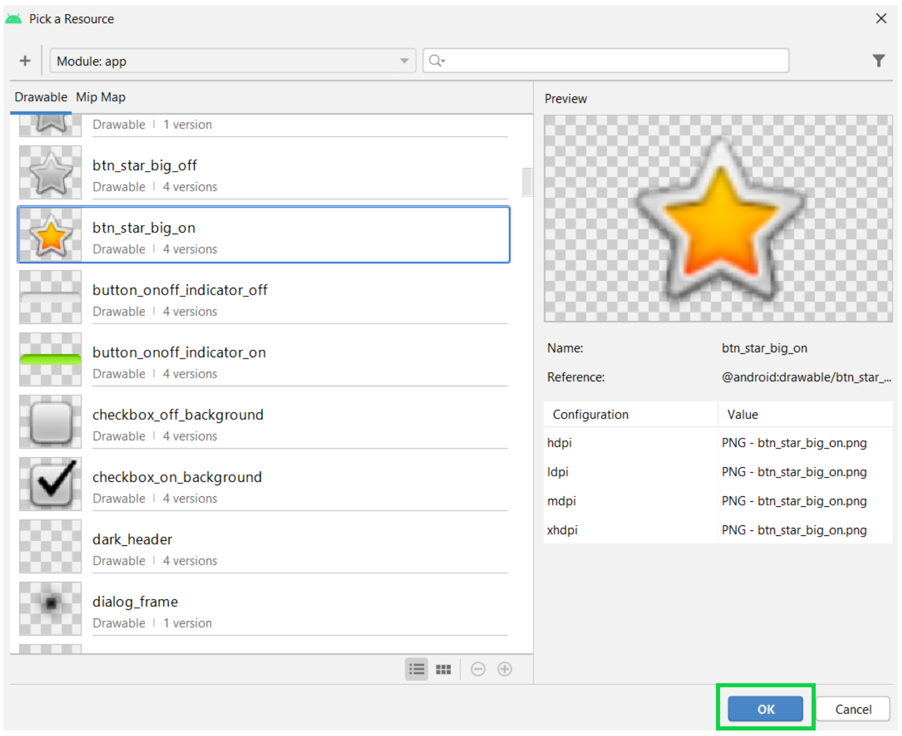
1. Open the activity_main.xml File in which the Image is to be Added
2. Switch from the Code View to the Design View of the activity_main.xml File



id	
layout_width	match_parent
layout_height	match_parent
context	.MainActivity
layout_width	match_parent
layout_height	match_parent
visibility	
visibility	
minWidth	
maxWidth	
minHeight	
maxHeight	
alpha	
accessibilityLiveR...	
actionBarNavMo...	
addStatesFromC...	
alpha	

3. For adding an image from Android Studio, Drag the ImageView widget to the activity area of the application, a pop-up dialogue box will open choose from the wide range of drawable resources and click “OK“.

For adding an image from Android Studio, Drag the ImageView widget to the activity area of the application, a pop-up dialogue box will open choose from the wide range of drawable resources and click “OK“.

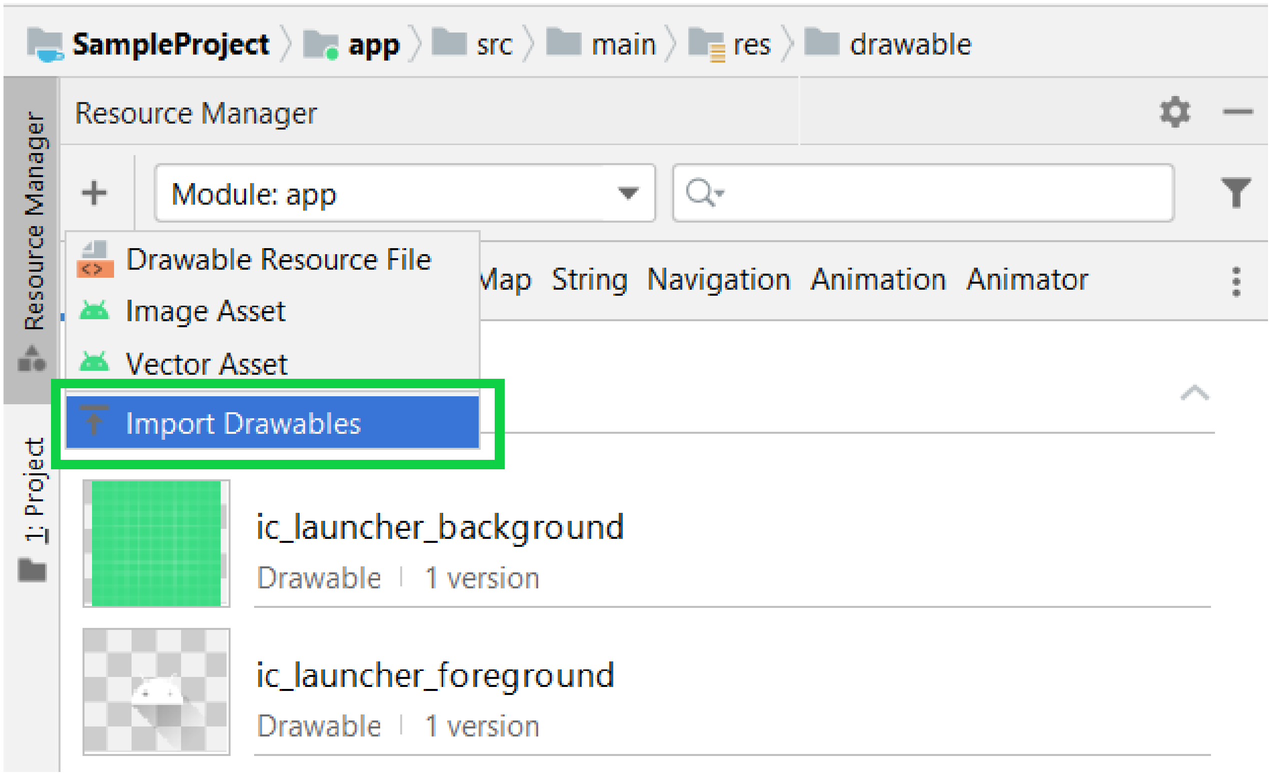


Drawable | 1 version
btn_star_big_off
Drawable | 4 versions
btn_star_big_on
Drawable | 4 versions
button_onoff_indicator_off
Drawable | 4 versions
button_onoff_indicator_on
Drawable | 4 versions
checkbox_off_background
Drawable | 4 versions
checkbox_on_background
Drawable | 4 versions
dark_header
Drawable | 4 versions
dialog_frame
Drawable | 1 version

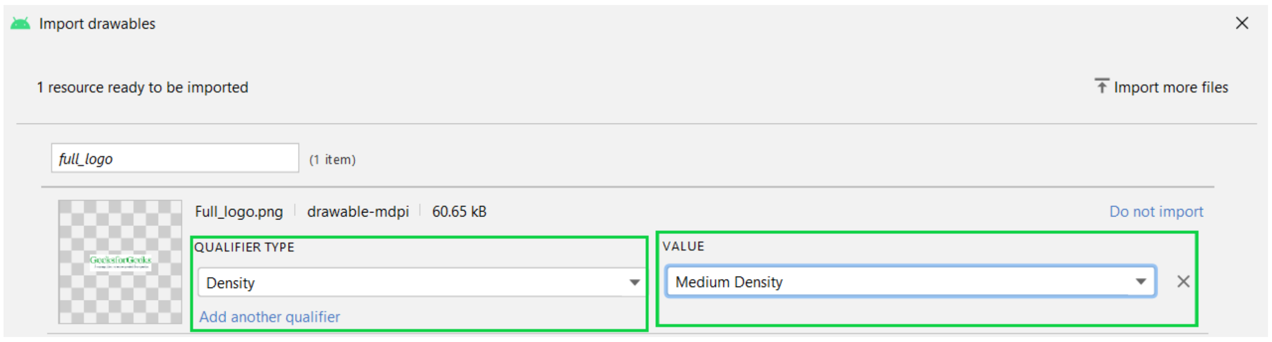
Name: btn_star_big_on
Reference: @android:drawable/btn_star_...

Configuration	Value
hdpi	PNG - btn_star_big_on.png
ldpi	PNG - btn_star_big_on.png
mdpi	PNG - btn_star_big_on.png
xhdpi	PNG - btn_star_big_on.png

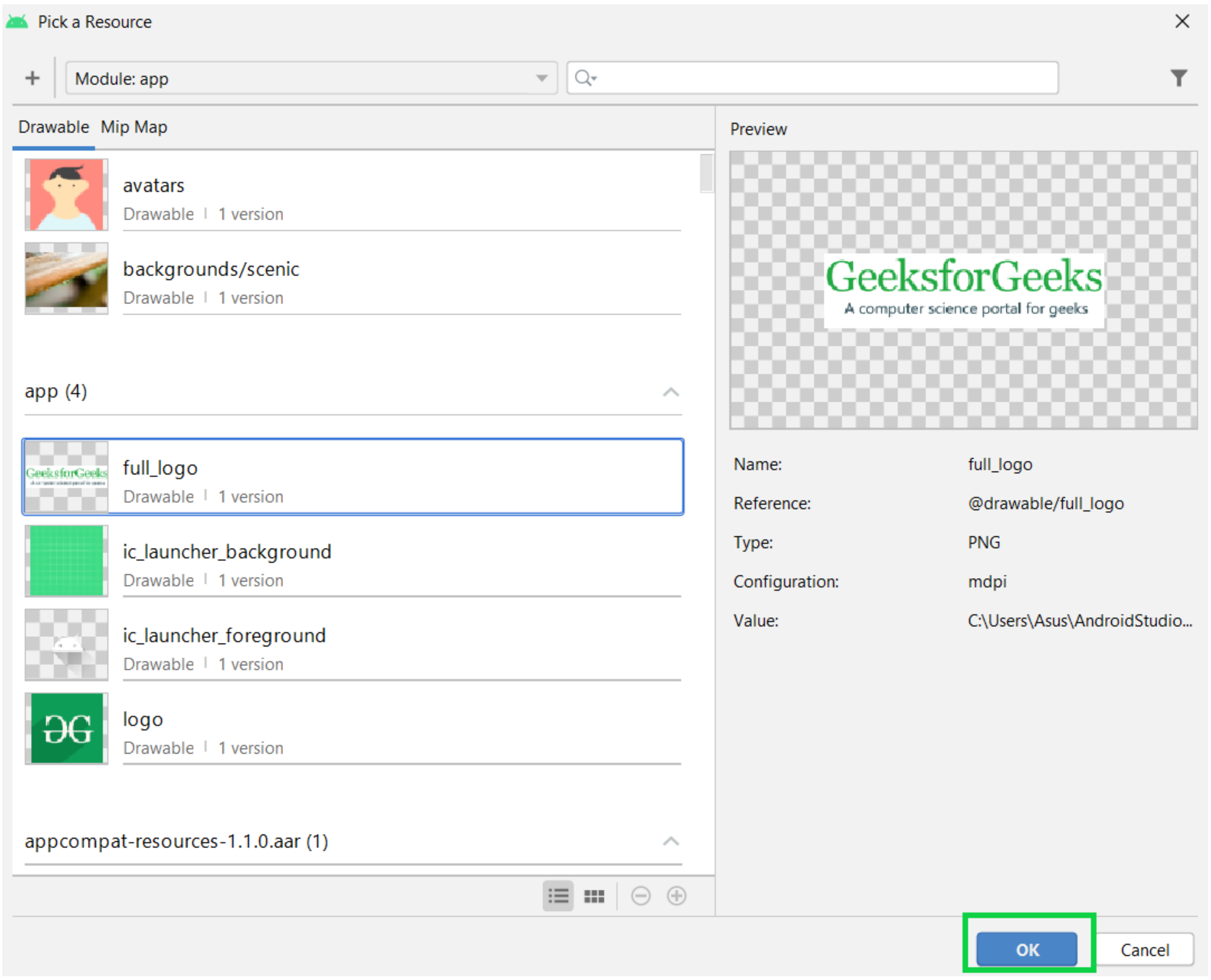
Click on the “**Resource Manager**” tab on the leftmost panel and select the “**Import Drawables**” option.



Select the path of the image file on your computer and click “**OK**“. After that set, the “**Qualifier type**” and “**value**” of the image file according to your need and click “**Next**” then “**Import**“.



Drag the ImageView class in the activity area, a pop-up dialogue box will appear which contains your imported image file. Choose your image file and click “**OK**“, your image will be added to the activity.



Name: full_logo
Reference: @drawable/full_logo
Type: PNG
Configuration: mdpi
Value: C:\Users\Asus\AndroidStudio...

XML Attributes of ImageView

XML Attribute	Description
android:id	To uniquely identify an image view
android:src/app:srcCompat	To add the file path of the inserted image
android:background	To provide a background color to the inserted image
android:layout_width	To set the width of the image

XML Attribute	Description
android:layout_height	To set the height of the image
android:padding	To add padding to the image from the left, right, top, or bottom of the view
android:scaleType	To re-size the image or to move it in order to fix its size

Step by Step Implementation

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio.

Step 2: Working with the activity_main.xml FileWorking with the activity_main.xml File

Go to the **activity_main.xml** File and refer to the following code. Below is the code for the **activity_main.xml** File.

Navigate to the **app > res > layout > activity_main.xml** and add the below code to that file. Below is the code for the **activity_main.xml** file.

XML

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

    xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">
    <ImageView
        android:id="@+id/GfG_full_logo"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.078"
        app:srcCompat="@drawable/full_logo" />

    <ImageView
        android:id="@+id/GfG_logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/GfG_full_logo"
        app:srcCompat="@drawable/logo" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

***Note**: All the attributes of the ImageView which are starting with app:layout_constraint are the vertical and horizontal constraints to fix the image position in the activity. This is very necessary to add the constraint to the ImageView otherwise, all the images will take the position (0, 0) of the activity layout.*

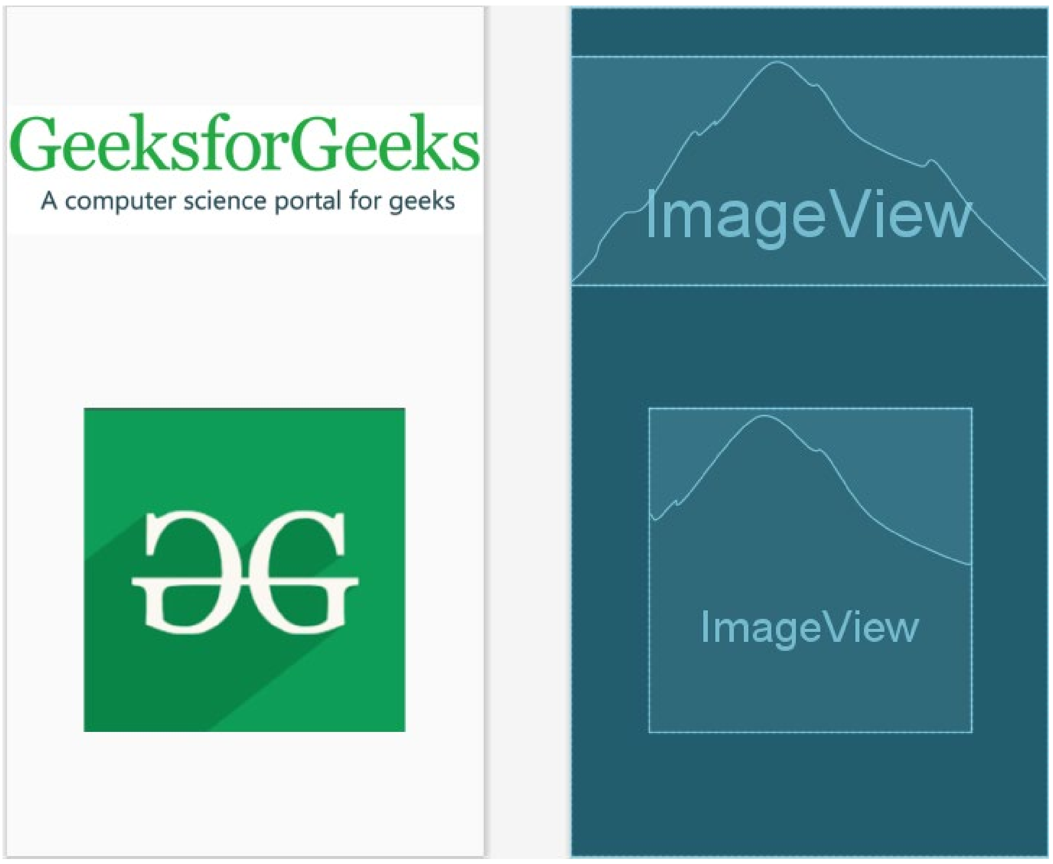
Step 4: Working with the MainActivity File

Go to the **MainActivity** file and refer to the following code. Below is the code for the **MainActivity** file. Since in the activity, only 2 images have been added and nothing else is being done like touching a button, etc. So, the **MainActivity** file will simply look like the below code i.e. no change.

Java

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;



public class MainActivity extends AppCompatActivity {



    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

    }

}
```

OUTPUT



Android - Image Switcher

- Sometimes you don't want an image to appear abruptly on the screen, rather you want to apply some kind of animation to the image when it transitions from one image to another.
- This is supported by android in the form of ImageSwitcher.
- An image switcher allows you to add some transitions on the images through the way they appear on screen.
- In order to use image Switcher, you need to define its XML component first.

Its syntax is given below –

```
     <ImageSwitcher
   android:id="@+id/imageSwitcher1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_centerHorizontal="true"
   android:layout_centerVertical="true" >
</ImageSwitcher>
```

- Now we create an intance of ImageSwithcer in java file and get a reference of this XML component.

Its syntax is given below –

```
private ImageSwitcher imageSwitcher;
imageSwitcher = (ImageSwitcher)findViewById(R.id.imageSwitcher1);
```

- The last thing you need to do is to add Animation to the ImageSwitcher. You need to define an object of Animation class through AnimationUtilities class by calling a static method loadAnimation.

Its syntax is given below −

```
Animation in =
AnimationUtils.loadAnimation(this,android.R.anim.slide_in_
left);
imageSwitcher.setInAnimation(in);
imageSwitcher.setOutAnimation(out);
```

Apart from these methods, there are other methods defined in the ***ImageSwitcher*** *class. They are defined below –*

1. Using Android Studio, create a new Android project and name it **ImageSwitcher**.
2. Modify the activity_main.xml file by adding the following bolded statements. Please be sure to change all instances of com.jfdimarzio to reflect the package you use in your project:

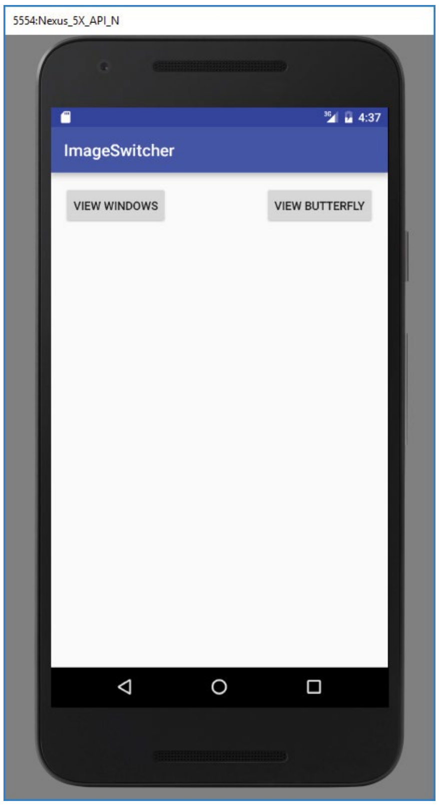
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.jfdimarzio.imageswitcher.MainActivity">
    <Button
        android:text="View Windows"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2"
        />

    <ImageSwitcher
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/button2"
        android:id="@+id/imageSwitcher">
    </ImageSwitcher>

    <Button
        android:text="View Butterfly"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"


         android:layout_alignParentTop="true"
         android:layout_alignParentEnd="true"
         />
 </RelativeLayout>
```

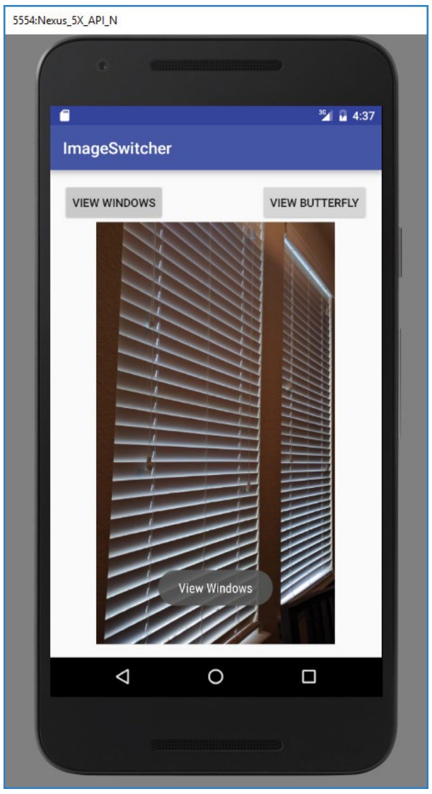
3. Add two images to your res/mipmap folder (as you did in the previous Try It Out). For this example, I added an image named butterfly.png and an image named windows.jpg.

4. Add the following bolded statements to the MainActivity.java file:

```
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.Toast;
import android.widget.ViewSwitcher;

public class MainActivity extends AppCompatActivity {
    private ImageSwitcher imgSwitcher;
    private Button btnViewWindows,btnViewButterfly;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        imgSwitcher = (ImageSwitcher) findViewById(R.id.imageSwitcher);
        imgSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_in));
        imgSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_out));


     , "View Windows",Toast.LENGTH_LONG).show();
                imgSwitcher.setImageResource(R.mipmap.windows);
            }
        });

        btnViewButterfly.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(), "View Butterfly"
     ,Toast.LENGTH_LONG).show();
                imgSwitcher.setImageResource(R.mipmap.butterfly);
            }
        });
    }
}
```

5. Press Shift+F9 to debug the application on the Android emulator. Figure 6-3 shows the ImageSwitcher and Button views, with no images loaded.





6. Click the View Windows button. You see the windows image and a Toast indicating that the image is being viewed,

GridView

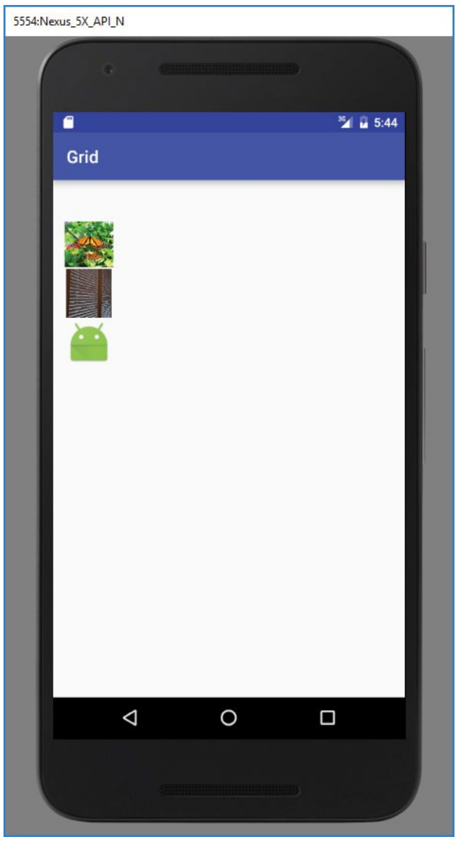
The `GridView` shows items in a two-dimensional scrolling grid. You can use the `GridView` together with an `ImageView` to display a series of images.

How It Works

Like the `ImageSwitcher` example, you first implement the `ImageAdapter` class and then bind it to the `GridView`:

```
 GridView gridView = (GridView) findViewById(R.id.gridview);
 gridView.setAdapter(new ImageAdapter(this));
 gridView.setOnItemClickListener(new OnItemClickListener()
 {
     public void onItemClick(AdapterView parent,
     View v, int position, long id)


     {
         Toast.makeText(getBaseContext(),
                 "pic" + (position + 1) + " selected",
                 Toast.LENGTH_SHORT).show();
     }
});
```

OUTPUT



Using menus With Views

Menus are useful for displaying additional options that are not directly visible on the main user interface (UI) of an application.
There are two main types of menus in Android:

Options menu—This menu displays information related to the current activity. In Android, you activate the options menu by pressing the Menu button.
Context menu—This menu displays information related to a particular view on an activity. In Android, you tap and hold a context menu to activate it.

Creating the helper methods

Before you go ahead and create your options and context menus, you need to create two helper methods. One creates a list of items to show inside a menu, whereas the other handles the event that is fired when the user selects an item inside the menu.

Creating the Menu helper Methods

1. Using Android Studio, create a new Android project and name it **Menus**.
2. In the `MainActivity.java` file, add the following bolded statements:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }



      private void createMenu(Menu menu) {
          MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
          {
              mnu1.setAlphabeticShortcut('a');
          }
          MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
          {
              mnu2.setAlphabeticShortcut('b');
          }
          MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
          {
              mnu3.setAlphabeticShortcut('c');
          }
          MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
          {
              mnu4.setAlphabeticShortcut('d');
          }
          menu.add(0, 4, 4, "Item 5");
          menu.add(0, 5, 5, "Item 6");
          menu.add(0, 6, 6, "Item 7");
      }
              case 4:
                  Toast.makeText(this, "You clicked on Item 5",
                          Toast.LENGTH_LONG).show();
                  return true;
              case 5:
                  Toast.makeText(this, "You clicked on Item 6",
                          Toast.LENGTH_LONG).show();
                  return true;
              case 6:
                  Toast.makeText(this, "You clicked on Item 7",
                          Toast.LENGTH_LONG).show();
                  return true;
          }
          return false;
      }
  }
```

How It Works
The preceding example creates two methods:

- `createMenu()`
- `menuChoice()`

The `createMenu()` method adds a series of menu items to a `Menu` argument.
To add a menu item to the menu, you create an instance of the `MenuItem` class and use the `Menu` object's `add()` method:

```
MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
{
    mnu1.setAlphabeticShortcut('a');
    mnu1.setIcon(R.mipmap.ic_launcher);
}
```

The four arguments of the `add()` method are

- `groupId`—The group identifier of which the menu item should be a part. Use 0 if an item is not in a group.
- `itemId`—A unique item ID.
- `order`—The order in which the item should be displayed.
- `title`—The text to display for the menu item.

You can use the setAlphabeticShortcut() method to assign a shortcut key to the menu item so that users can select an item by pressing a key on the keyboard. The setIcon() method sets an image to be displayed on the menu item.

The menuChoice() method takes a MenuItem argument and checks its ID to determine the menu item that is selected. It then displays a Toast message to let the user know which menu item was selected.

Options Menu

You are now ready to modify the application to display the options menu when the user presses the Menu key on the Android device.

Using the same project created in the previous section, add the following bolded statements to the `MainActivity.java` file:

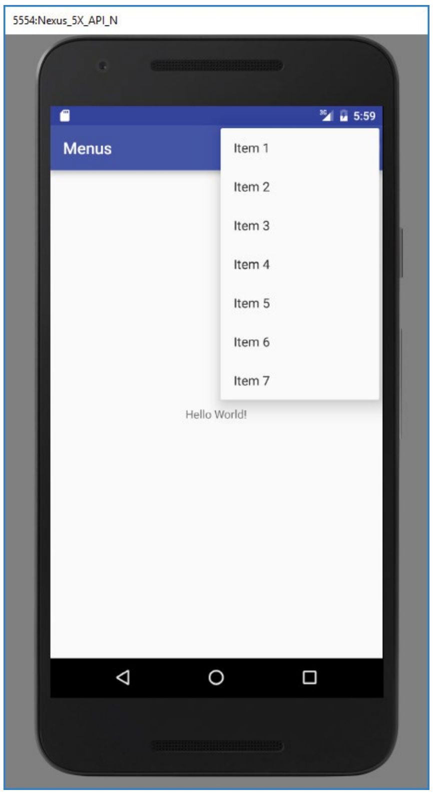
1.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        createMenu(menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        return menuChoice(item);
    }


    private void createMenu(Menu menu) {
        MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
        {
            mnu1.setAlphabeticShortcut('a');
        }
        MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
        {
            mnu2.setAlphabeticShortcut('b');
        }
        MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
        {
            mnu3.setAlphabeticShortcut('c');
        }
        MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
        {
            mnu4.setAlphabeticShortcut('d');
        }
        menu.add(0, 4, 4, "Item 5");
        menu.add(0, 5, 5, "Item 6");
        menu.add(0, 6, 6, "Item 7");
    }

    private boolean menuChoice(MenuItem item) {
        switch (item.getItemId()) {
            case 0:
                Toast.makeText(this, "You clicked on Item 1",
                        Toast.LENGTH_LONG).show();


    return true;
case 1:
    Toast.makeText(this, "You clicked on Item 2",
            Toast.LENGTH_LONG).show();
    return true;
```

```
                case 2:
                    Toast.makeText(this, "You clicked on Item 3",
                            Toast.LENGTH_LONG).show();
                    return true;
                case 3:
                    Toast.makeText(this, "You clicked on Item 4",
                            Toast.LENGTH_LONG).show();
                    return true;
                case 4:
                    Toast.makeText(this, "You clicked on Item 5",
                            Toast.LENGTH_LONG).show();
                    return true;
                case 5:
                    Toast.makeText(this, "You clicked on Item 6",
                            Toast.LENGTH_LONG).show();
                    return true;
                case 6:
                    Toast.makeText(this, "You clicked on Item 7",
                            Toast.LENGTH_LONG).show();
                    return true;
            }
            return false;
        }
    }
```

2. Press Shift+F9 to debug the application on the Android emulator. Figure 6-7 shows the options menu that displays when you click the Menu button. To select a menu item, either click an individual item or use its shortcut key



How It Works

To display the options menu for your activity, you need to implement two methods in your activity:

- `onCreateOptionsMenu()`
- `onOptionsItemSelected()`

The `onCreateOptionsMenu()` method is called when the Menu button is pressed. In this case, you call the `createMenu()` helper method to display the options menu.
When a menu item is selected, the `onOptionsItemSelected()` method is called. In this case, you call the `menuChoice()` method to display the menu item selected (and perform whatever action is appropriate).

<u>Context menu</u>

- A context menu is usually associated with a view on an activity. A context menu is displayed when the user taps and holds an item.
- For example, if the user taps a `Button` view and holds it for a few seconds, a context menu can be displayed.
- If you want to associate a context menu with a view on an activity, you need to call the **setOnCreateContextMenuListener()** method of that particular view.

1. Using the same project from the previous example, add the following bolded statements to the activity_main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.jfdimarzio.menus.MainActivity"
    tools:layout_editor_absoluteX="0dp"
    tools:layout_editor_absoluteY="81dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        tools:layout_editor_absoluteX="154dp"
        tools:layout_editor_absoluteY="247dp"
        app:layout_constraintLeft_toLeftOf="@+id/activity_main"
        tools:layout_constraintLeft_creator="0"
        app:layout_constraintTop_toTopOf="@+id/activity_main"
        tools:layout_constraintTop_creator="0"
        app:layout_constraintRight_toRightOf="@+id/activity_main"
        tools:layout_constraintRight_creator="0"
        app:layout_constraintBottom_toBottomOf="@+id/activity_main"
        tools:layout_constraintBottom_creator="0" />
    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:layout_editor_absoluteX="148dp"
        tools:layout_editor_absoluteY="102dp"
        android:id="@+id/button"
        app:layout_constraintLeft_toLeftOf="@+id/activity_main"
        tools:layout_constraintLeft_creator="0"
        app:layout_constraintRight_toRightOf="@+id/activity_main"
        tools:layout_constraintRight_creator="0" />

</android.support.constraint.ConstraintLayout>
```

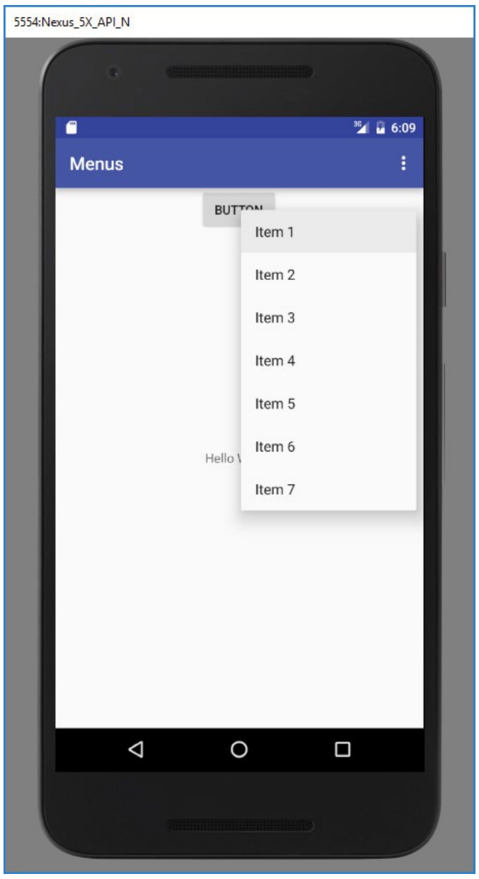
3. Add the following bolded statements to the MenusActivity.java file:

```
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button btn = (Button) findViewById(R.id.button);
        btn.setOnCreateContextMenuListener(this);
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View view,
                                    ContextMenu.ContextMenuInfo menuInfo)
    {
        super.onCreateContextMenu(menu, view, menuInfo);
        createMenu(menu);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        createMenu(menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        return menuChoice(item);
    }
```

```
private void createMenu(Menu menu) {
    MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
    {
        mnu1.setAlphabeticShortcut('a');
    }
    MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
    {
        mnu2.setAlphabeticShortcut('b');



       }
       MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
       {
           mnu3.setAlphabeticShortcut('c');
       }
       MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
       {
           mnu4.setAlphabeticShortcut('d');
       }
       menu.add(0, 4, 4, "Item 5");
       menu.add(0, 5, 5, "Item 6");
       menu.add(0, 6, 6, "Item 7");
   }

   private boolean menuChoice(MenuItem item) {
       switch (item.getItemId()) {
           case 0:
               Toast.makeText(this, "You clicked on Item 1",
                       Toast.LENGTH_LONG).show();
               return true;
           case 1:
               Toast.makeText(this, "You clicked on Item 2",
                       Toast.LENGTH_LONG).show();
               return true;
           case 2:
               Toast.makeText(this, "You clicked on Item 3",
                       Toast.LENGTH_LONG).show();
               return true;
           case 3:
               Toast.makeText(this, "You clicked on Item 4",
                       Toast.LENGTH_LONG).show();
               return true;
      case 4:
          Toast.makeText(this, "You clicked on Item 5",
                  Toast.LENGTH_LONG).show();
          return true;
      case 5:
          Toast.makeText(this, "You clicked on Item 6",
                  Toast.LENGTH_LONG).show();
          return true;
      case 6:
          Toast.makeText(this, "You clicked on Item 7",
                  Toast.LENGTH_LONG).show();
          return true;
```

```
                }
                return false;
            }
        }
```

4. Press Shift+F9 to debug the application on the Android emulator. Figure 6-8 shows the context menu that displays when you click and hold the Button view.

5. 

Android - WebView

WebView is a view that display web pages inside your application. You can also specify HTML string and can show it inside your application using WebView. WebView makes turns your application to a web application.

In order to add WebView to your application, you have to add **<WebView>** element to your xml layout file. Its syntax is as follows −

```
<WebView
xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/webview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
/>
```

In order to use it, you have to get a reference of this view in Java file. To get a reference, create an object of the class WebView. Its syntax is −

WebView browser = (WebView) findViewById(R.id.webview);

In order to load a web url into the WebView, you need to call a method **loadUrl(String url)** of the WebView class, specifying the required url. Its syntax is:

browser.loadUrl("https://www.tutorialspoint.com");

Apart from just loading url, you can have more control over your WebView by using the methods defined in WebView class. They are listed as follows −

Sr.No	Method & Description
1	**canGoBack()** This method specifies the WebView has a back history item.
2	**canGoForward()** This method specifies the WebView has a forward history item.
3	**clearHistory()** This method will clear the WebView forward and backward history.
4	**destroy()** This method destroy the internal state of WebView.
5	**findAllAsync(String find)** This method find all instances of string and highlight them.
6	**getProgress()** This method gets the progress of the current page.
7	**getTitle()** This method return the title of the current page.
8	**getUrl()** This method return the url of the current page.

If you click on any link inside the webpage of the WebView, that page will not be loaded inside your WebView. In order to do that you need to extend your class from **WebViewClient** and override its method. Its syntax is −

```
private class MyBrowser extends WebViewClient {
   @Override
   public boolean shouldOverrideUrlLoading(WebView view,
String url) {
        view.loadUrl(url);
        return true;
    }
}
```

Example

Here is an example demonstrating the use of WebView Layout. It creates a basic web application that will ask you to specify a url and will load this url website in the WebView.

To experiment with this example, you need to run this on an actual device on which internet is running.

Steps	**Description**
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add WebView code.
3	Modify the res/layout/activity_main to add respective XML components
4	Modify the AndroidManifest.xml to add the necessary permissions
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file src/MainActivity.java.

```
package com.example.sairamkrishna.myapplication;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity  {
   Button b1;
   EditText ed1;

   private WebView wv1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      b1=(Button)findViewById(R.id.button);
      ed1=(EditText)findViewById(R.id.editText);

      wv1=(WebView)findViewById(R.id.webView);
      wv1.setWebViewClient(new MyBrowser());

      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            String url = ed1.getText().toString();

            wv1.getSettings().setLoadsImagesAutomatically(true);
            wv1.getSettings().setJavaScriptEnabled(true);
            wv1.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);
            wv1.loadUrl(url);
         }
      });
   }
   private class MyBrowser extends WebViewClient {
      @Override
      public boolean shouldOverrideUrlLoading(WebView view, String url) {
         view.loadUrl(url);
         return true;
      }
   }
}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
   android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

   <TextView android:text="WebView"
android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/textview"
      android:textSize="35dp"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true" />

   <TextView
```

```
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point"
      android:id="@+id/textView"
      android:layout_below="@+id/textview"
      android:layout_centerHorizontal="true"
      android:textColor="#ff7aff24"
      android:textSize="35dp" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText"
      android:hint="Enter Text"
      android:focusable="true"
      android:textColorHighlight="#ff7eff15"
      android:textColorHint="#ffff25e6"
      android:layout_marginTop="46dp"
      android:layout_below="@+id/imageView"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true"
      android:layout_alignRight="@+id/imageView"
      android:layout_alignEnd="@+id/imageView" />

   <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageView"
      android:src="@drawable/abc"
      android:layout_below="@+id/textView"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Enter"
      android:id="@+id/button"
      android:layout_alignTop="@+id/editText"
      android:layout_toRightOf="@+id/imageView"
      android:layout_toEndOf="@+id/imageView" />

   <WebView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/webView"
      android:layout_below="@+id/button"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true"
      android:layout_alignParentRight="true"
      android:layout_alignParentEnd="true"
      android:layout_alignParentBottom="true" />
```

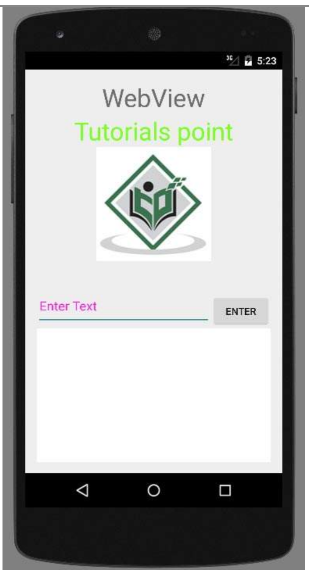
```
</RelativeLayout>
```

Following is the content of the **res/values/string.xml.**

```
<resources>
   <string name="app_name">My Application</string>
</resources>
```

Following is the content of **AndroidManifest.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.sairamkrishna.myapplication" >
   <uses-permission android:name="android.permission.INTERNET" />
   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >

      <activity
         android:name=".MainActivity"
         android:label="@string/app_name" >

         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
android:name="android.intent.category.LAUNCHER" />
         </intent-filter>

      </activity>

   </application>
</manifest>
```



Saving and loading user preferences

Android provides the *SharedPreferences* object to help you save simple application data. For example, your application may have an option that enables users to specify the font size used in your application. In this case, your application needs to remember the size set by the user so that the size is set appropriately each time the app is opened. You have several options for saving this type of preference:

➤➤ **Save data to a file**—You can save the data to a file, but you have to perform some file management routines, such as writing the data to the file, indicating how many characters to read from it, and so on. Also, if you have several pieces of information to save, such as text size, font name, preferred background color, and so on, then the task of writing to a file becomes more onerous.

➤➤ **Writing text to a database**—An alternative to writing to a text file is to use a database.However, saving simple data to a database is overkill, both from a developer's point of view and in terms of the application's run-time performance.

➤➤ **Using the SharedPreferences object**—The SharedPreferences object, however, saves data through the use of name/value pairs. For example, specify a name for the data you want to save, and then both it and its value will be saved automatically to an XML file.

Android - Shared Preferences

- Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of ***key,value*** pair.
- **Shared Preferences** is the way in which one can store and retrieve small amounts of primitive data as key/value pairs to a file on the device storage such as String, int, float, Boolean that make up your preferences in an XML file inside the app on the device storage.
- For example, you might have a key being “username” and for the value, you might store the user’s username. And then you could retrieve that by its key (here username).
- You can have a simple shared preference API that you can use to store preferences and pull them back as and when needed.
- The shared Preferences class provides APIs for reading, writing, and managing this data.
- In order to use shared preferences, you have to call a method **getSharedPreferences()** that returns a SharedPreference instance pointing to the file that contains the values of preferences.

SharedPreferences sharedpreferences = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);

The first parameter is the key and the second parameter is the MODE.

This method takes two arguments, the first being the name of the SharedPreference(SP) file and the other is the context mode that we want to store our file in.

- *MODE_PUBLIC* will make the file public which could be accessible by other applications on the device

- *MODE_PRIVATE* keeps the files private and secures the user’s data.

- *MODE_APPEND* is used while reading the data from the SharedPreference file.

Following are the methods of Shared Preferences

1. **contains(String key)**: This method is used to check whether the preferences contain a preference.
2. **edit()**: This method is used to create a new Editor for these preferences, through which you can make modifications to the data in the preferences and atomically commit those changes back to the SharedPreferences object.
3. **getAll()**: This method is used to retrieve all values from the preferences.
4. **getBoolean(String key, boolean defValue)**: This method is used to retrieve a boolean value from the preferences.
5. **getFloat(String key, float defValue)**: This method is used to retrieve a float value from the preferences.

6. **getInt(String key, int defValue)**: This method is used to retrieve an int value from the preferences.
7. **getLong(String key, long defValue)**: This method is used to retrieve a long value from the preferences.
8. **getString(String key, String defValue)**: This method is used to retrieve a String value from the preferences.
9. **getStringSet(String key, Set defValues)**: This method is used to retrieve a set of String values from the preferences.
10. **registerOnSharedPreferencechangeListener(SharedPreferences.OnSharedPreferencechangeListener listener)**: This method is used to register a callback to be invoked when a change happens to a preference.
11. **unregisterOnSharedPreferencechangeListener(SharedPreferences.OnSharedPreferencechangeListener listener)**: This method is used to unregister a previous callback.

You can save something in the sharedpreferences by using ***SharedPreferences.Editor*** class. You will call the edit method of SharedPreference instance and will receive it in an editor object. Its syntax is –

```
Editor editor = sharedpreferences.edit();
editor.putString("key", "value");
editor.commit();
```

Apart from the ***putString*** method , there are methods available in the editor class that allows manipulation of data inside shared preferences. They are listed as follows –

Sr. NO	Mode & description
1	**apply()** It is an abstract method. It will commit your changes back from editor to the sharedPreference object you are calling
2	**clear()** It will remove all values from the editor
3	**remove(String key)** It will remove the value whose key has been passed as a parameter
4	**putLong(String key, long value)** It will save a long value in a preference editor
5	**putInt(String key, int value)** It will save a integer value in a preference editor
6	**putFloat(String key, float value)** It will save a float value in a preference editor

Example
This example demonstrates the use of the Shared Preferences. It display a screen with some text fields, whose value are saved when the application is closed and brought back when it is opened again.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add progress code to display the spinning progress dialog.
3	Modify res/layout/activity_main.xml file to add respective XML code.
4	Run the application and choose a running android device and install the application on it and verify the results.

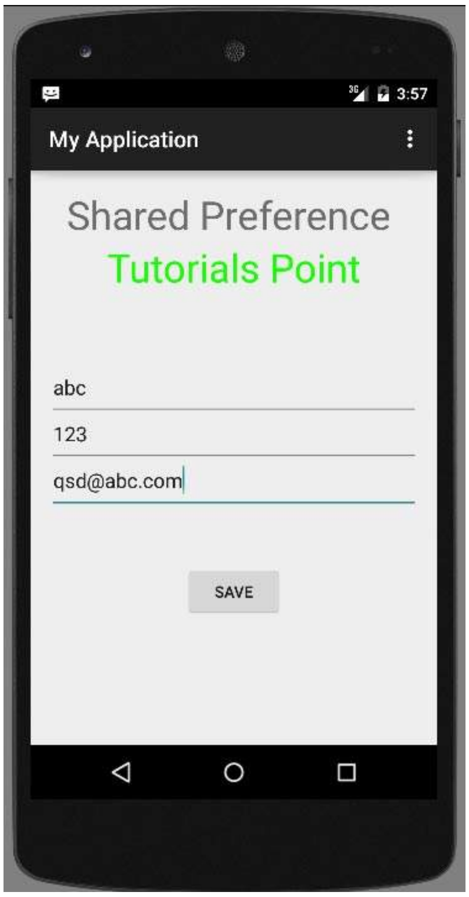
Following is the content of the modified **MainActivity.java.**

```java
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   EditText ed1,ed2,ed3;
   Button b1;

   public static final String MyPREFERENCES = "MyPrefs" ;
   public static final String Name = "nameKey";
   public static final String Phone = "phoneKey";
   public static final String Email = "emailKey";

   SharedPreferences sharedpreferences;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      ed1=(EditText)findViewById(R.id.editText);
      ed2=(EditText)findViewById(R.id.editText2);
      ed3=(EditText)findViewById(R.id.editText3);

      b1=(Button)findViewById(R.id.button);
      sharedpreferences =
getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);

      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            String n  = ed1.getText().toString();
            String ph  = ed2.getText().toString();
            String e  = ed3.getText().toString();

            SharedPreferences.Editor editor =
sharedpreferences.edit();

            editor.putString(Name, n);
            editor.putString(Phone, ph);
            editor.putString(Email, e);
            editor.commit();

Toast.makeText(MainActivity.this,"Thanks",Toast.LENGTH_LON
G).show();
         }
      });
   }

}
```

Following is the content of the modified main activity file**res/layout/activiy_main.xml.**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
   android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Shared Preference "
      android:id="@+id/textView"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="35dp" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials Point"
      android:id="@+id/textView2"
      android:layout_below="@+id/textView"
      android:layout_centerHorizontal="true"
      android:textSize="35dp"
      android:textColor="#ff16ff01" />
   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText"
      android:layout_below="@+id/textView2"
      android:layout_marginTop="67dp"
      android:hint="Name"
      android:layout_alignParentRight="true"
      android:layout_alignParentEnd="true"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText2"
      android:layout_below="@+id/editText"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true"
      android:layout_alignParentRight="true"
      android:layout_alignParentEnd="true"
      android:hint="Pass" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText3"
      android:layout_below="@+id/editText2"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true"
      android:layout_alignParentRight="true"
      android:layout_alignParentEnd="true"
      android:hint="Email" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Save"
      android:id="@+id/button"
      android:layout_below="@+id/editText3"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="50dp" />

</RelativeLayout>
```

Now just put in some text in the field. Like i put some random name and other information and click on save button.



abc

123

qsd@abc.com

Now when you press save button, the text will be saved in the shared preferences. Now press back button and exit the application. Now open it again and you will see all the text you have written back in your application.

Android supports the following ways of storing data in the local file system:

- Files - You can create and update files
- Preferences - Android allows you to save and retrieve persistent key-value pairs of primitive data type.
- SQLite database - instances of SQLite databases are also stored on the local file system.

Android - Internal Storage

- Android provides many kinds of storage for applications to store their data. These storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection.

- In this chapter we are going to look at the internal storage. Internal storage is the storage of the private data on the device memory.

- By default these files are private and are accessed by only your application and get deleted , when user delete your application.

Writing file

- In order to use internal storage to write some data in the file, call the *openFileOutput()* method with the name of the file and the mode. The mode could be private , public e.t.c.

- Its syntax is given below –

```
FileOutputStream fOut = openFileOutput("file name
here",MODE_WORLD_READABLE);
```

The method **openFileOutput()** returns an instance of ***FileOutputStream***. So you receive it in the object of ***FileInputStream***. After that you can call write method to write data on the file.

Its syntax is given below –

```
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

Reading file

- In order to read from the file you just created , call the ***openFileInput()*** method with the name of the file. It returns an instance of FileInputStream.

- Its syntax is given below –

```
FileInputStream fin = openFileInput(file);
```

- After that, you can call read method to read one character at a time from the file and then you can print it.

Its syntax is given below –

```
int c;
String temp="";
while( (c = fin.read()) != -1){
   temp = temp + Character.toString((char)c);
}

//string temp contains all the data of the file.
fin.close();
```

Apart from the methods of write and close, there are other methods provided by the ***FileOutputStream*** class for better writing files.

These methods are listed below –

Sr.No	Method & description
1	**FileOutputStream(File file, boolean append)** This method constructs a new FileOutputStream that writes to file.
2	**getChannel()** This method returns a write-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**write(byte[] buffer, int byteOffset, int byteCount)** This method Writes count bytes from the byte array buffer starting at position offset to this stream

Apart from the the methods of read and close, there are other methods provided by the **FileInputStream** class for better reading files. These methods are listed below –

Sr.No	Method & description
1	**available()** This method returns an estimated number of bytes that can be read or skipped without blocking for more input
2	**getChannel()** This method returns a read-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**read(byte[] buffer, int byteOffset, int byteCount)** This method reads at most length bytes from this stream and stores them in the byte array b starting at offset

Example
Here is an example demonstrating the use of internal storage to store and read files. It creates a basic storage application that allows you to read and write from internal storage.

Steps	Description
1	You will use Android Studio IDE to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components
4	Run the application and choose a running android device and install the application on it and verify the results

Following is the content of the modified main activity file **src/MainActivity.java**.

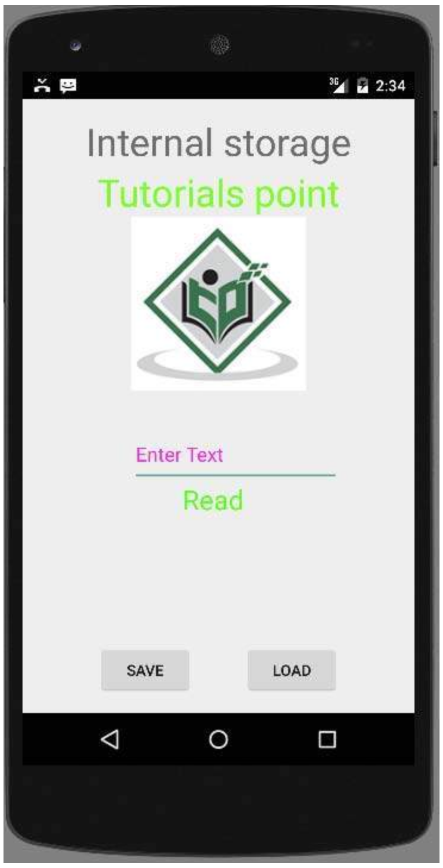
```
package com.example.sairamkrishna.myapplication;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
```

```java
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class MainActivity extends Activity  {
   Button b1,b2;
   TextView tv;
   EditText ed1;

   String data;
   private String file = "mydata";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      b1=(Button)findViewById(R.id.button);
      b2=(Button)findViewById(R.id.button2);

      ed1=(EditText)findViewById(R.id.editText);
      tv=(TextView)findViewById(R.id.textView2);
      b1.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            data=ed1.getText().toString();
            try {
               FileOutputStream fOut =
openFileOutput(file,MODE_WORLD_READABLE);
               fOut.write(data.getBytes());
               fOut.close();
               Toast.makeText(getBaseContext(),"file
saved",Toast.LENGTH_SHORT).show();
            }
            catch (Exception e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
            }
         }
      });

      b2.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            try {
               FileInputStream fin = openFileInput(file);
               int c;
               String temp="";
               while( (c = fin.read()) != -1){
                  temp = temp +
Character.toString((char)c);
               }
               tv.setText(temp);
               Toast.makeText(getBaseContext(),"file
read",Toast.LENGTH_SHORT).show();
            }
            catch(Exception e){
            }
         }
      });
   }
}
```

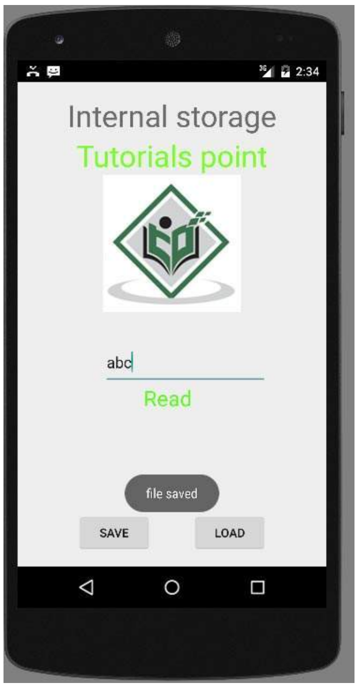
Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
   android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

   <TextView android:text="Internal storage"
android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/textview"
      android:textSize="35dp"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true" />

   <TextView
```

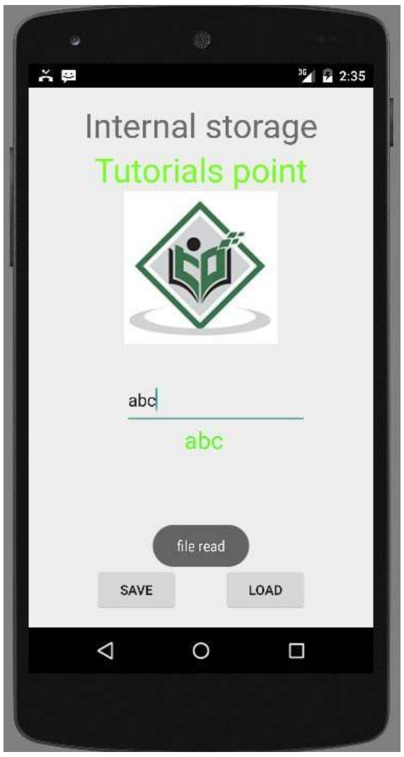
```xml
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point"
      android:id="@+id/textView"
      android:layout_below="@+id/textview"
      android:layout_centerHorizontal="true"
      android:textColor="#ff7aff24"
      android:textSize="35dp" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Save"
      android:id="@+id/button"
      android:layout_alignParentBottom="true"
      android:layout_alignLeft="@+id/textView"
      android:layout_alignStart="@+id/textView" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText"
      android:hint="Enter Text"
      android:focusable="true"
      android:textColorHighlight="#ff7eff15"
      android:textColorHint="#ffff25e6"
      android:layout_below="@+id/imageView"
      android:layout_alignRight="@+id/textView"
      android:layout_alignEnd="@+id/textView"
      android:layout_marginTop="42dp"
      android:layout_alignLeft="@+id/imageView"
      android:layout_alignStart="@+id/imageView" />

   <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageView"
      android:src="@drawable/abc"
      android:layout_below="@+id/textView"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="load"
      android:id="@+id/button2"
      android:layout_alignTop="@+id/button"
      android:layout_alignRight="@+id/editText"
      android:layout_alignEnd="@+id/editText" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Read"
      android:id="@+id/textView2"
      android:layout_below="@+id/editText"
      android:layout_toLeftOf="@+id/button2"
      android:layout_toStartOf="@+id/button2"
      android:textColor="#ff5bff1f"
      android:textSize="25dp" />

</RelativeLayout>
```



Now what you need to do is to enter any text in the field. For example , i have entered some text. Press the save button. The following notification would appear in you AVD –



Now when you press the load button, the application will read the file , and display the data.



Persisting Data to Files

The ***SharedPreferences object*** enables you to store data that is best stored as name/value pairs— for example, user ID, birth date, gender, driver's license number, and so on. However, sometimes you might prefer to use the traditional file system to store your data. For example, you might want to store the text of poems you want to display in your applications. In Android, you can use the classes in the **java.io package** to do so.

saving to internal storage

The first way to save files in your Android application is to write to the device's internal storage. The following Try It Out demonstrates how to save a string entered by the user to the device's internal storage.

1. Using Android Studio, create an Android project and name it **Files**.

2. In the `activity_main.xml` file, add the following bolded statements:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.jfdimarzio.files.MainActivity">

    <TextView
        android:text="Please enter some text."
        android:layout_width="245dp"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        app:layout_constraintLeft_toLeftOf="@+id/activity_main"
       app:layout_constraintTop_toTopOf="@+id/activity_main"
        android:layout_marginTop="16dp"
        app:layout_constraintRight_toRightOf="@+id/activity_main"
        app:layout_constraintBottom_toTopOf="@+id/editText"
        android:layout_marginBottom="8dp"
        app:layout_constraintVertical_bias="0.28" />

    <EditText
        android:layout_width="241dp"
        android:layout_height="wrap_content"
        android:inputType="text"
        android:ems="10"
        tools:layout_editor_absoluteY="82dp"
        android:id="@+id/editText"
        app:layout_constraintLeft_toLeftOf="@+id/activity_main"
        app:layout_constraintRight_toRightOf="@+id/activity_main"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        android:layout_marginTop="136dp"/>

    <Button
        android:text="Save"
        android:layout_width="240dp"
        android:layout_height="wrap_content"
        android:id="@+id/btnSave"
        app:layout_constraintLeft_toLeftOf="@+id/activity_main"
        android:layout_marginStart="16dp"
        app:layout_constraintTop_toBottomOf="@+id/editText"
        android:layout_marginTop="136dp"
        app:layout_constraintRight_toRightOf="@+id/activity_main"
        android:layout_marginEnd="16dp"
        android:onClick="onClickSave" />

    <Button
        android:text="Load"
        android:layout_width="241dp"
        android:layout_height="wrap_content"
        android:id="@+id/btnLoad"
        app:layout_constraintLeft_toLeftOf="@+id/activity_main"
        android:layout_marginStart="16dp"
        app:layout_constraintTop_toBottomOf="@+id/editText"
        android:layout_marginTop="48dp"
        app:layout_constraintRight_toRightOf="@+id/activity_main"
        android:layout_marginEnd="16dp"
        android:onClick="onClickLoad" />

</android.support.constraint.ConstraintLayout>
```

3. In the `MainActivity.java` file, add the following bolded statements:

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class MainActivity extends AppCompatActivity {
    EditText textBox;
    static final int READ_BLOCK_SIZE = 100;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textBox = (EditText) findViewById(R.id.editText);
    }

    public void onClickSave(View view) {
        String str = textBox.getText().toString();
        try {
            FileOutputStream fOut = openFileOutput("textfile.txt",
MODE_PRIVATE);
            OutputStreamWriter osw = new OutputStreamWriter(fOut);
            //---write the string to the file---
            try {
                osw.write(str);
            } catch (IOException e) {
                e.printStackTrace();
            }
            osw.flush();
            osw.close();
            //---display file saved message---
            Toast.makeText(getBaseContext(),
      "File saved successfully!", Toast.LENGTH_SHORT).show();
            //---clears the EditText---
            textBox.setText("");
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }

    public void onClickLoad(View view) {
        try {
            FileInputStream fIn = openFileInput("textfile.txt");
            InputStreamReader isr = new InputStreamReader(fIn);
            char[] inputBuffer = new char[READ_BLOCK_SIZE];
            String s = "";
            int charRead;
            while ((charRead = isr.read(inputBuffer)) > 0) {
                //---convert the chars to a String---
                String readString =
                        String.copyValueOf(inputBuffer, 0,
                                charRead);
                s += readString;
                inputBuffer = new char[READ_BLOCK_SIZE];
            }
            //---set the EditText to the text that has been
            // read---
            textBox.setText(s);
            Toast.makeText(getBaseContext(), "File loaded successfully!",
                    Toast.LENGTH_SHORT).show();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

4. Press Shift+F9 to debug the application on the Android emulator.
5. Type some text into the ***EditText view*** (see Figure 7-9) and then click the Save button.

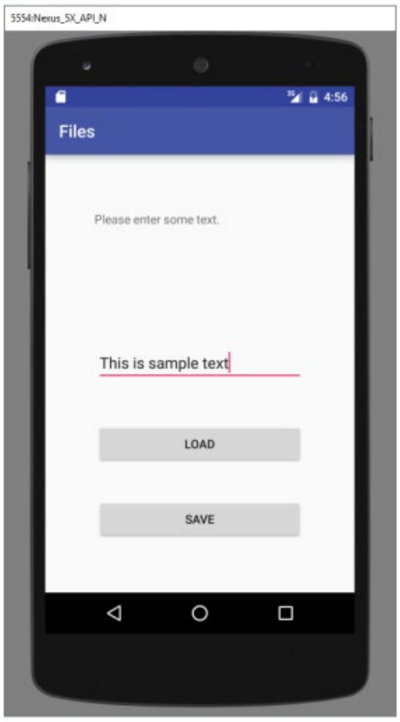


FIGURE 7-9

6. If the file is saved successfully, you see the Toast class displaying the "File saved successfully!"message. The text in the ***EditText view*** should disappear.

7. Click the Load button and you should see the string appearing in the EditText view again. This confirms that the text is saved correctly.

How It Works

To save text into a file, you use the **`FileOutputStream`** class. The **`openFileOutput()`** method opens a named file for writing, with the mode specified. In this example, you use the `MODE_PRIVATE` constant to indicate that the file is readable by all other applications:

```
FileOutputStream fOut = openFileOutput("textfile.txt", MODE_PRIVATE);
```

To convert a character stream into a byte stream, you use an instance of the **`OutputStreamWriter`** class, by passing it an instance of the **`FileOutputStream`** object:

```
OutputStreamWriter osw = new OutputStreamWriter(fOut);
```

You then use its **`write()`** method to write the string to the file. To ensure that all the bytes are written to the file, use the **`flush()`** method. Finally, use the **`close()`** method to close the file:

```
//---write the string to the file--- osw.write(str);
osw.flush(); osw.close();
```

7. To read the content of a file, you use the **`FileInputStream`** class, together with the

```
InputStreamReader class: FileInputStream fIn = openFileInput("textfile.txt");
InputStreamReader isr = new InputStreamReader(fIn);
```

Because you do not know the size of the file to read, the content is read in blocks of 100 characters into a buffer (character array). The characters read are then copied into a `String` object:

```
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";
int charRead;
while ((charRead = isr.read(inputBuffer))>0)
       {
//---convert the chars to a String---
String readString = String.copyValueOf(inputBuffer, 0, charRead); s += readString;
inputBuffer = new char[READ_BLOCK_SIZE];
       }
```

The **`read()`** method of the **`InputStreamReader`** object checks the number of characters read and returns –1 if the end of the file is reached.

Saving to External storage (sd Card)

Sometimes, it would be useful to save them to external storage (such as an SD card) because of its larger capacity, as well as the capability to share the files easily with other users (by removing the SD card and passing it to somebody else).
You can use the following steps to save files to external storage.

1. Using the project created in the previous section as the example (saving text entered by the user to the SD card), modify the `onClick()` method of the Save button as shown in bold here:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
    public void onClickSave(View view) {
         String str = textBox.getText().toString();
         try
         {
             //---SD Card Storage---
             File sdCard = Environment.getExternalStorageDirectory();
             File directory = new File (sdCard.getAbsolutePath() +
                 "/MyFiles");
             directory.mkdirs();
             File file = new File(directory, "textfile.txt");
             FileOutputStream fOut = new FileOutputStream(file);
             /*
             FileOutputStream fOut =
                     openFileOutput("textfile.txt",
                             MODE_WORLD_READABLE);
             */

             OutputStreamWriter osw = new
                     OutputStreamWriter(fOut);
             //---write the string to the file---
             osw.write(str);
             osw.flush();
             osw.close();
             //---display file saved message---
             Toast.makeText(getBaseContext(),
                     "File saved successfully!",
                     Toast.LENGTH_SHORT).show();
             //---clears the EditText---
             textBox.setText("");
         }
         catch (IOException ioe)
         {
             ioe.printStackTrace();
         }
    }
```

2. The preceding code uses the **`getExternalStorageDirectory()`** method to return the full path to the external storage. Typically, it should return the "**`/sdcard`**" path for a real device, and "**`/mnt/sdcard`**" for an Android emulator. However, you should never try to hardcode the path to the SD card, as manufacturers may choose to assign a different path name to the SD card. Be sure to use the **`getExternalStorageDirectory()`** method to return the full path to the SD card.

3. You then create a directory called **`MyFiles`** in the SD card.

4. Finally, you save the file into this directory.

5. To load the file from the external storage, modify the **`onClickLoad()`** method for the Load button:

```
public void onClickLoad(View view) {
    try
    {
        //---SD Storage---
        File sdCard = Environment.getExternalStorageDirectory();
        File directory = new File (sdCard.getAbsolutePath() +
            "/MyFiles");
        File file = new File(directory, "textfile.txt");
        FileInputStream fIn = new FileInputStream(file);
        InputStreamReader isr = new InputStreamReader(fIn);
        /*
        FileInputStream fIn =
                openFileInput("textfile.txt");
        InputStreamReader isr = new
                InputStreamReader(fIn);
        */

        char[] inputBuffer = new char[READ_BLOCK_SIZE];
        String s = "";
        int charRead;
        while ((charRead = isr.read(inputBuffer))>0)
        {
            //---convert the chars to a String---
            String readString =
                    String.copyValueOf(inputBuffer, 0,
                            charRead);
            s += readString;
            inputBuffer = new char[READ_BLOCK_SIZE];
        }
        //---set the EditText to the text that has been
        // read---
        textBox.setText(s);
        Toast.makeText(getBaseContext(),
                "File loaded successfully!",



                    Toast.LENGTH_SHORT).show();
      }
      catch (IOException ioe) {
          ioe.printStackTrace();
      }
  }
```

6. Note that in order to write to the external storage, you need to add the **`WRITE_EXTERNAL_STORAGE`** permission in your **`AndroidManifest.xml`** file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.jfdimarzio.Files"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name=
"android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".FilesActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Choosing the Best storage option

Which one should you use in your applications? Here are some guidelines:

- If you have data that can be represented using name/value pairs, then use the `SharedPreferences` object. For example, if you want to store user preference data such as username, background color, date of birth, or last login date, then the `SharedPreferences` object is the ideal way to store this data. Moreover, you don't really have to do much to store data this way. Simply use the `SharedPreferences` object to store and retrieve it.

- If you need to store ad-hoc data then using the internal storage is a good option. For example, your application (such as an RSS reader) might need to download images from the web for display. In this scenario, saving the images to internal storage is a good solution. You might also need to persist data created by the user, such as when you have an application that enables users to take notes and save them for later use. In both of these scenarios, using the internal storage is a good choice.

- There are times when you need to share your application data with other users. For example ,you might create an Android application that logs the coordinates of the locations that a user has been to, and subsequently, you want to share all this data with other users. In this scenario, you can store your files on the SD card of the device so that users can easily transfer the data to other devices (and computers) for use later.

Creating and using dataBases

- So far, all the techniques you have seen are useful for saving simple sets of data.
- For saving relational data, using a database is much more efficient.
- For example, if you want to store the test results of all the students in a school, it is much more efficient to use a database to represent them because you can use database querying to retrieve the results of specific students.
- Moreover, using databases enables you to enforce data integrity by specifying the relationships between different sets of data.
- Android uses the SQLite database system.
- The database that you create for an application is only accessible to itself; other applications will not be able to access it.
- In this section, you find out how to programmatically create a **SQLite database** in your Android application.

For Android, the SQLite database that you create programmatically in an application is always stored in the `/data/data/<package_name>/databases` folder.

Creating the dBadapter helper Class

1. Using Android Studio, create an Android project and name it **Databases**.
2. Add a new Java Class file to the package and name it **DBAdapter**

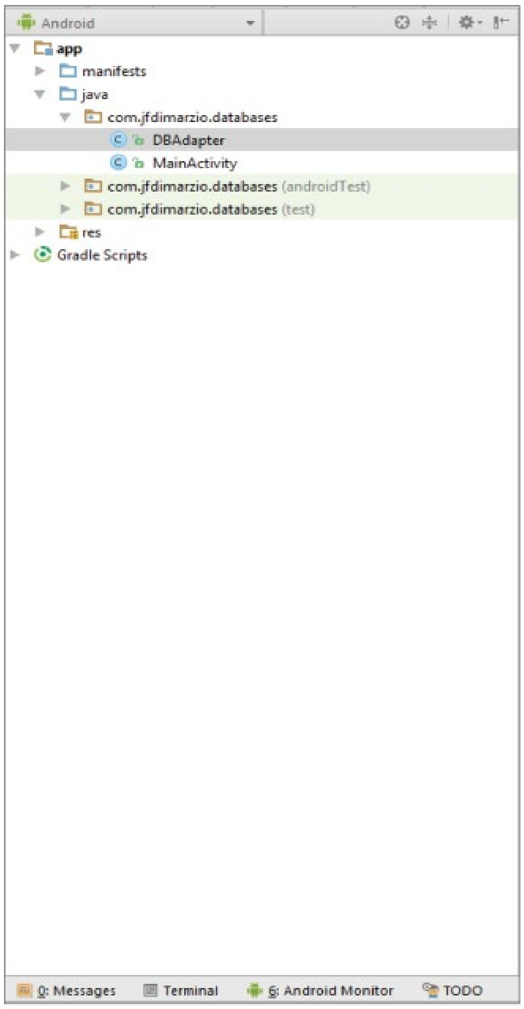


FIGURE 7-10

3. Add the following bolded statements to the **DBAdapter.java** file:

```
import android.content.ContentValues;
import android.content.Context;
```

```
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DBAdapter {
    static final String KEY_ROWID = "_id";
    static final String KEY_NAME = "name";
    static final String KEY_EMAIL = "email";
    static final String TAG = "DBAdapter";
    static final String DATABASE_NAME = "MyDB";
    static final String DATABASE_TABLE = "contacts";
    static final int DATABASE_VERSION = 1;
    static final String DATABASE_CREATE =
            "create table contacts (_id integer primary key autoincrement, "
                    + "name text not null, email text not null);";
    final Context context;
    DatabaseHelper DBHelper;
    SQLiteDatabase db;

    public DBAdapter(Context ctx)
    {
        this.context = ctx;
        DBHelper = new DatabaseHelper(context);
    }
    private static class DatabaseHelper extends SQLiteOpenHelper
    {
        DatabaseHelper(Context context)
        {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db)
        {
            try {
                db.execSQL(DATABASE_CREATE);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
        {
            Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                    + newVersion + ", which will destroy all old data");
            db.execSQL("DROP TABLE IF EXISTS contacts");
            onCreate(db);
        }
    }
    //---opens the database---
    public DBAdapter open() throws SQLException
```

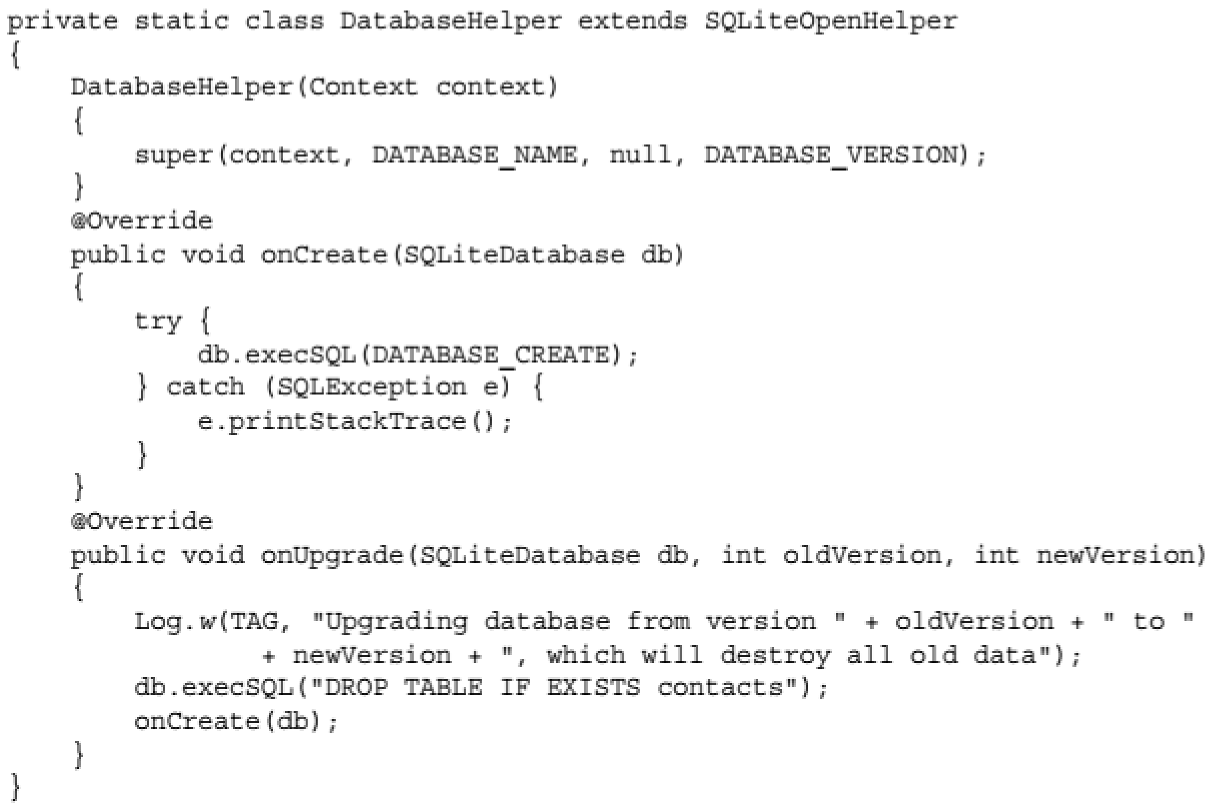
```
        {
            db = DBHelper.getWritableDatabase();
            return this;
        }
        //---closes the database---
        public void close()
        {
            DBHelper.close();
        }
        //---insert a contact into the database---
        public long insertContact(String name, String email)
        {
            ContentValues initialValues = new ContentValues();
            initialValues.put(KEY_NAME, name);
            initialValues.put(KEY_EMAIL, email);
            return db.insert(DATABASE_TABLE, null, initialValues);
        }
        //---deletes a particular contact---
        public boolean deleteContact(long rowId)
        {
            return db.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
        }
        //---retrieves all the contacts---
        public Cursor getAllContacts()
        {
            return db.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
                    KEY_EMAIL}, null, null, null, null, null);
        }
        //---retrieves a particular contact---
        public Cursor getContact(long rowId) throws SQLException
        {
            Cursor mCursor =
                    db.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
                                    KEY_NAME, KEY_EMAIL},
         KEY_ROWID + "=" + rowId, null,
                            null, null, null, null);
            if (mCursor != null) {
                mCursor.moveToFirst();
            }
            return mCursor;
        }
        //---updates a contact---
        public boolean updateContact(long rowId, String name, String email)
        {
            ContentValues args = new ContentValues();
            args.put(KEY_NAME, name);
            args.put(KEY_EMAIL, email);
            return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) >
    0;
        }
    }
```

How It Works

You first define several constants to contain the various fields for the table that you are going to create in your database:

```
static final String KEY_ROWID = "_id";
static final String KEY_NAME = "name";
static final String KEY_EMAIL = "email";
static final String TAG = "DBAdapter";
static final String DATABASE_NAME = "MyDB";
static final String DATABASE_TABLE = "contacts"; static final int DATABASE_VERSION = 1;
static final String DATABASE_CREATE =
"create table contacts (_id integer primary key autoincrement, " + "name text not null,
email text not null);";
```

In particular, the **`DATABASE_CREATE`** constant contains the SQL statement for creating the **`contacts`** table within the **`MyDB`** database.
Within the **`DBAdapter`** class, you also add a private class that extends the **`SQLiteOpenHelper`** class. **`SQLiteOpenHelper`** is a helper class in Android to manage database creation and version management. In particular, you must override the **`onCreate()`** and **`onUpgrade()`** methods:



```
private static class DatabaseHelper extends SQLiteOpenHelper
{
    DatabaseHelper(Context context)
    {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db)
    {
        try {
            db.execSQL(DATABASE_CREATE);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS contacts");
        onCreate(db);
    }
}
```

The **`onCreate()`** method creates a new database if the required database is not present. The **`onUpgrade()`** method is called when the database needs to be upgraded. This is achieved by checking the value defined in the **`DATABASE_VERSION`** constant. For this implementation of the `onUpgrade()` method, you simply drop the table and create it again.

You can then define the various methods for opening and closing the database, as well as the methods for adding/editing/deleting rows in the table:

```
//---opens the database---
public DBAdapter open() throws SQLException
{
    db = DBHelper.getWritableDatabase();
    return this;
}
//---closes the database---
public void close()
{
    DBHelper.close();
}
//---insert a contact into the database---
public long insertContact(String name, String email)
{
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_NAME, name);
    initialValues.put(KEY_EMAIL, email);
    return db.insert(DATABASE_TABLE, null, initialValues);
}
//---deletes a particular contact---
public boolean deleteContact(long rowId)
{
    return db.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}
//---retrieves all the contacts---
public Cursor getAllContacts()
{
    return db.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
            KEY_EMAIL}, null, null, null, null, null);
}
//---retrieves a particular contact---
public Cursor getContact(long rowId) throws SQLException
{
    Cursor mCursor =
            db.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
            KEY_NAME, KEY_EMAIL}, KEY_ROWID + "=" + rowId, null,
            null, null, null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}
//---updates a contact---
public boolean updateContact(long rowId, String name, String email)
{
    ContentValues args = new ContentValues();
    args.put(KEY_NAME, name);
    args.put(KEY_EMAIL, email);
    return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
```

Notice that Android uses the **`Cursor`** class as a return value for queries. Think of the `Cursor` as a pointer to the result set from a database query. Using `Cursor` enables Android to more efficiently manage rows and columns as needed.

You use a **`ContentValues`** object to store name/value pairs. Its **`put()`** method enables you to insert keys with values of different data types.

To create a database in your application using the **`DBAdapter`** class, you create an instance of the `DBAdapter` class:

```
      public DBAdapter(Context ctx)

             {

             this.context = ctx; DBHelper = new DatabaseHelper(context);

              }
```

The constructor of the `DBAdapter` class will then create an instance of the `DatabaseHelper` class to create a new database:

```
      DatabaseHelper(Context context)

            {

                 super(context, DATABASE_NAME, null, DATABASE_VERSION);

             }
```

Using the Database Programmatically

With the **`DBAdapter`** helper class created, you are now ready to use the database. In the following sections, you will learn how to perform the regular **CRUD (create, read, update and delete)** operations commonly associated with databases.

Adding Contacts

The following Try It Out demonstrates how you can add a contact to the table

1. Using the same project created earlier, add the following bolded statements to the `MainActivity.java` file:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        DBAdapter db = new DBAdapter(this);


        //---add a contact---
        db.open();
        long id = db.insertContact("Jennifer Ann",
"jenniferann@jfdimarzio.com");
        id = db.insertContact("Oscar Diggs", "oscar@oscardiggs.com");
        db.close();
    }
}
```

2. Press Shift+F9 to debug the application on the Android emulator.

How It Works
In this example, you create an instance of the `DBAdapter` class:

```
DBAdapter db = new DBAdapter(this);
```

The `insertContact()` method returns the ID of the inserted row. If an error occurs during the operation, it returns –1.

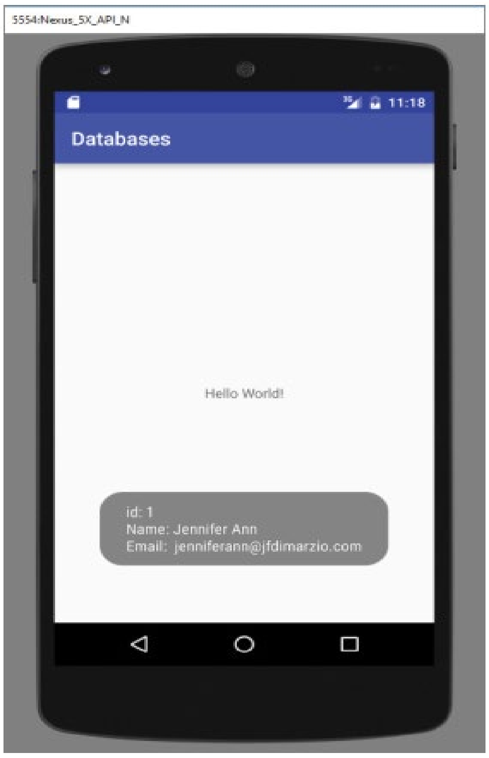
Retrieving all the Contacts

To retrieve all the contacts in the `contacts` table, use the **`getAllContacts()`** method of the `DBAdapter` class, as the following Try It Out shows.

1. Using the same project created earlier, add the following bolded statements to the `MainActivity.java` file:

```
import android.database.Cursor;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        DBAdapter db = new DBAdapter(this);
        //---add a contact---
        db.open();
        long id = db.insertContact("Jennifer Ann", "jenniferann@jfdimarzio.com");
        id = db.insertContact("Oscar Diggs", "oscar@oscardiggs.com");
        db.close();
        db.open();
         Cursor c = db.getAllContacts();
         if (c.moveToFirst())


      {
          do {
              DisplayContact(c);
          } while (c.moveToNext());
      }
      db.close();
  }

  public void DisplayContact(Cursor c)
  {
      Toast.makeText(this,
              "id: " + c.getString(0) + "\n" +
      "Name: " + c.getString(1) + "\n" +
      "Email:  " + c.getString(2),
          Toast.LENGTH_LONG).show();
  }
}
```

2. Press Shift+F9 to debug the application on the Android emulator. Figure 7-11 shows the `Toast` class displaying the contacts retrieved from the database.



id: 1
Name: Jennifer Ann
Email: jenniferann@jfdimarzio.com

FIGURE 7-11

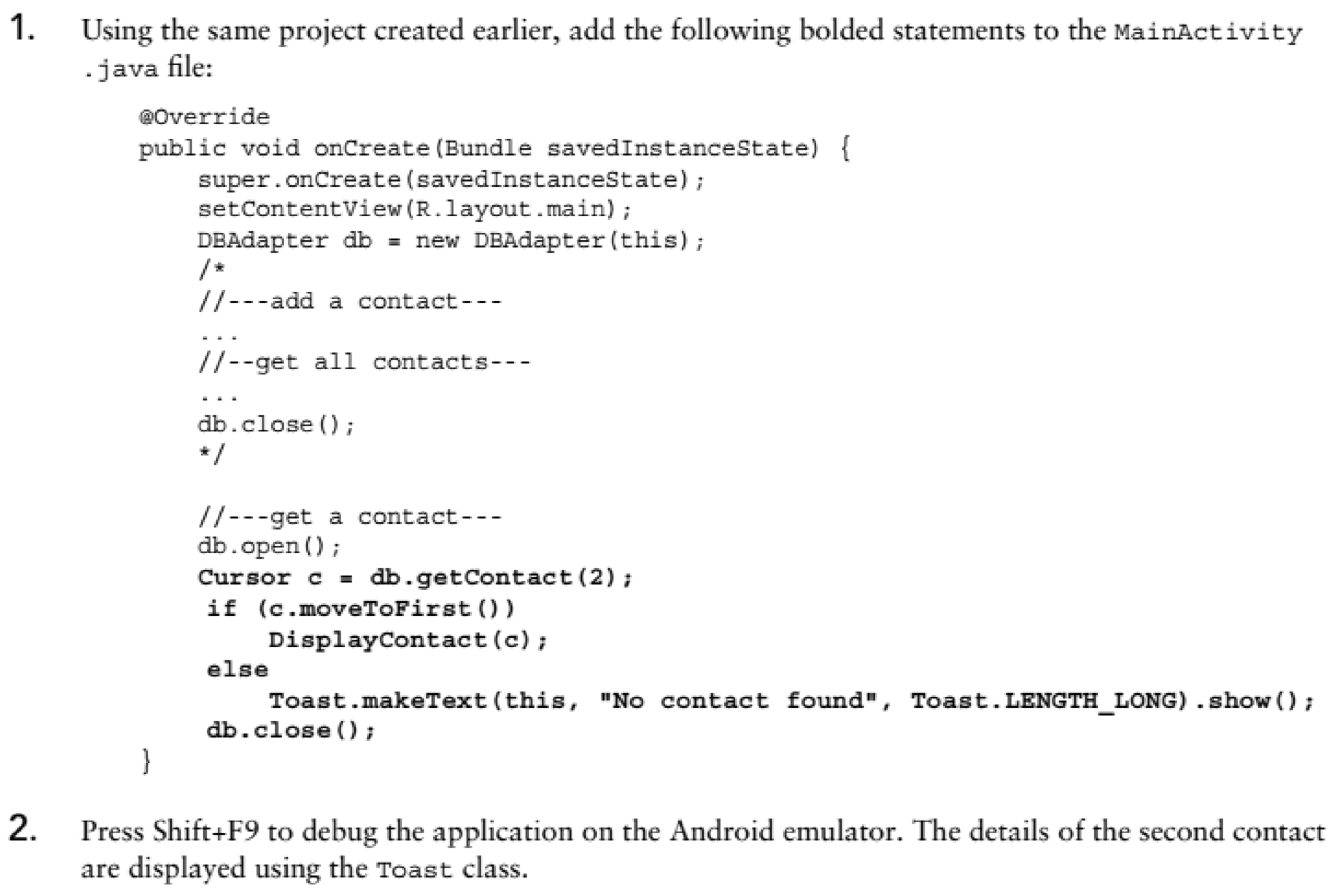
How It Works

The **`getAllContacts()`** method of the **`DBAdapter`** class retrieves all the contacts stored in the database. The result is returned as a **`Cursor`** object. To display all the contacts, you first need to call the **`moveToFirst()`** method of the `Cursor` object. If it succeeds (which means at least one row is available), then you display the details of the contact using the **`DisplayContact()`** method. To move to the next contact, call the **`moveToNext()`** method of the `Cursor` object.

Retrieving a Single Contact

To retrieve a single contact using its ID, call the **`getContact()`** method of the `DBAdapter` class, as the following Try It Out shows.

Retrieving a Contact from a table



1. Using the same project created earlier, add the following bolded statements to the `MainActivity.java` file:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    DBAdapter db = new DBAdapter(this);
    /*
    //---add a contact---
    ...
    //--get all contacts---
    ...
    db.close();
    */

    //---get a contact---
    db.open();
    Cursor c = db.getContact(2);
     if (c.moveToFirst())
         DisplayContact(c);
     else
         Toast.makeText(this, "No contact found", Toast.LENGTH_LONG).show();
     db.close();
}
```

2. Press Shift+F9 to debug the application on the Android emulator. The details of the second contact are displayed using the `Toast` class.

How It Works

The **`getContact()`** method of the `DBAdapter` class retrieves a single contact using its ID. You pass in the ID of the contact. In this case, you pass in an ID of 2 to indicate that you want to retrieve the second contact:

```
Cursor c = db.getContact(2);
```

Updating a Contact

To update a particular contact, call the **updateContact()** method in the DBAdapter class by passing the ID of the contact you want to update, as the following Try It Out shows.

Updating a Contact in a table

1. Using the same project created earlier, add the following bolded statements to the MainActivity .java file:

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        DBAdapter db = new DBAdapter(this);
        /*
        //---add a contact---
        ...
        //--get all contacts---
        ...
        //---get a contact---
        ...
        db.close();
        */

        //---update contact---
        db.open();
         if (db.updateContact(1, "Oscar Diggs", "oscar@oscardiggs.com"))
             Toast.makeText(this, "Update successful.",
Toast.LENGTH_LONG).show();
         else
             Toast.makeText(this, "Update failed.", Toast.LENGTH_LONG).show();
         db.close();
    }
```

2. Press Shift+F9 to debug the application on the Android emulator. A message is displayed if the update is successful.

How It Works

The **updateContact()** method in the DBAdapter class updates a contact's details by using the ID of the contact you want to update. It returns a Boolean value, indicating whether the update was successful.

Deleting a Contact

To delete a contact, use the `deleteContact()` method in the `DBAdapter` class by passing the ID of the contact you want to update, as the following Try It Out shows.

1. Using the same project created earlier, add the following bolded statements to the `MainActivity.java` file:

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        DBAdapter db = new DBAdapter(this);
         /*
         //---add a contact---
         ...
         //--get all contacts---
         ...
         //---get a contact---
         ...
         //---update contact---
         ...
         db.close();
         */

         //---delete a contact---
         db.open();
         if (db.deleteContact(1))
             Toast.makeText(this, "Delete successful.",
Toast.LENGTH_LONG).show();
         else
             Toast.makeText(this, "Delete failed.", Toast.LENGTH_LONG).show();
         db.close();
    }
```

2. Press Shift+F9 to debug the application on the Android emulator. A message is displayed if the deletion was successful.

How It Works

The **`deleteContact()`** method in the `DBAdapter` class deletes a contact using the ID of the contact you want to delete. It returns a Boolean value, indicating whether the deletion was successful.

Upgrading the Database

Sometimes, after creating and using the database, you might need to add additional tables, change the schema of the database, or add columns to your tables. In this case, you need to migrate your existing data from the old database to a newer one.
To upgrade the database, change the **`DATABASE_VERSION`** constant to a value higher than the previous one. For example, if its previous value was 1, change it to 2:

```
public class DBAdapter {
    static final String KEY_ROWID = "_id";
    static final String KEY_NAME = "name";
    static final String KEY_EMAIL = "email";

static final String TAG = "DBAdapter";
static final String DATABASE_NAME = "MyDB";
static final String DATABASE_TABLE = "contacts";
static final int DATABASE_VERSION = 2;
```

When you run the application one more time, you see the following message in the logcat window of Android Studio:

```
DBAdapter(8705): Upgrading database from version 1 to 2,
```