Programming in Android

Session: 5

More UI Elements

Objectives

- Explain the use of adapters
- Identify the different types of adapters
- Describe the advanced and complex UI Components
- Explain the use of Custom Dialogs
- Explain and create Custom Widgets
- Explain and use Material Design Philosophy

Introduction

- An Adapter is an object that acts as a bridge between the UI Components and the underlying data source
- Dialogs are interactive popup windows
- App Widgets are miniature application views that can be embedded in other applications and receive periodic updates
- Material Design is Google's new UI Design philosophy

Adapters

- An Adapter is an object that acts as a bridge between the UI Components such as List view, Grid view, and so on and the underlying data source
- The underlying data source can be an Array, database, and so on that fills data to the UI component
- Adapter is responsible for providing view for every element of the data source

Types of Adapters 1-5

- **BaseAdapter**
  - BaseAdapter is a common implementation for the adapter which can be used for both ListView and Spinner
  - It is an abstract base class for the adapter interface
  - The developer can implement his/her own adapters using BaseAdapter

- **SimpleAdapter**
  - A SimpleAdapter helps to map static data to the views defined in the XML
  - The data can be mapped to the view by using ArrayList of Maps and each element in the array will be represented as a separate row in the view

Types of Adapters 2-5

- **ArrayAdapter**
  - ArrayAdapter is backed by an array of objects to load the data to the UI view
  - In other words, it is used to bind an array of data to the view
  - It overrides the getView() method to inflate, populate, and return a custom view for the provided array data

- **SimpleCursorAdapter**
  - An adapter which maps columns from a cursor to TextViews or ImageViews defined in an XML file
  - The developer can specify which columns to be displayed, in which views the developer wants to display the columns, and the XML file that defines the appearance of these views

Types of Adapters 3-5

- **CursorAdapter**
  - This Adapter that is used for exposing the data from a Cursor to a ListView object using the column named _id

- **ResourceCursorAdapter**
  - ResourceCursorAdapter is very similar to the CursorAdapter, which doesn't have a new View method and is used for simple views
  - It is used to create views defined in an XML file. This Adapter is deprecated in API Level 11

Types of Adapters 4-5

- **SpinnerAdapter**
  - SpinnerAdapter acts as a bridge between the spinner component and the data source
  - SpinnerAdapter allows displaying of data in the following two ways:
    - Displays data in the spinner view
    - Displays data in the drop-down list when the spinner is pressed

- **SimpleCursorTreeAdapter**
  - This Adapter can be used to map column data from the Cursor to the TextView or ImageView as defined in the XML file
  - The developer can specify the required columns and separate child views to display the specified content. Binding is done using the setViewValue() method of the SimpleCursorTreeAdapter, ViewBinder
  - The method returns a boolean value which can be used to determine whether binding has been successful

Types of Adapters 5-5

- **CursorTreeAdapter**
  - This Adapter can be used to display data in an expandable list view by using the data from the cursor
  - In this, the top-level Cursor exposes the group and the getChildrenCursor(Cursor) method returns cursors which exposes the child elements within the particular group
- **HeaderViewListAdapter**
  - HeaderViewListAdapter can be used to implement a ListView having a header at the top
- **WrapperListAdapter**
  - WrapperListAdapter can be used to wrap another ListAdapter
  - The method, getWrappedAdapter() is invoked for retrieving the wrapped adapter

Implementation Using BaseAdapter 1-2

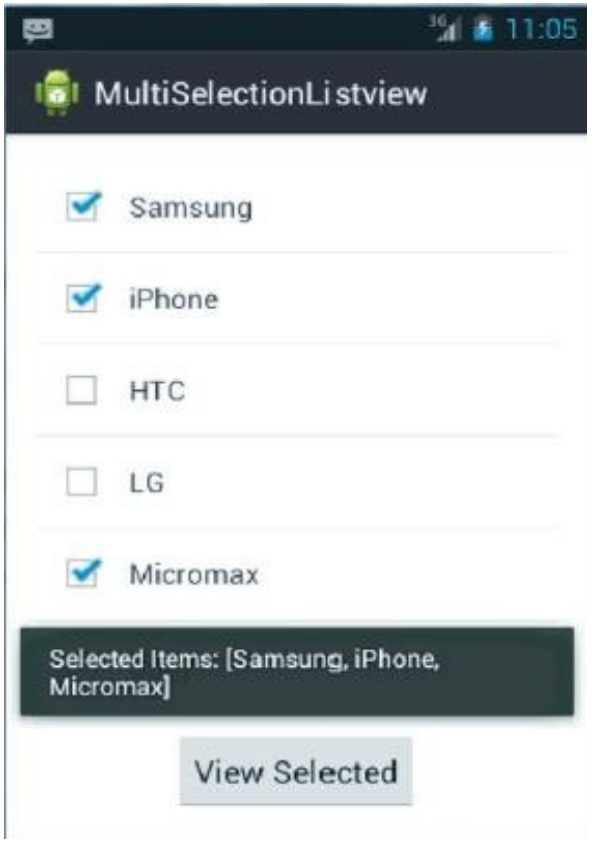
- The process for implementing a simple adapter using the BaseAdapter class is shown in the following Code Snippet:

```
public class MultiSelectionAdapter<T>
extends BaseAdapter {

  public MultiSelectionAdapter(Context
context,
ArrayList<T> list) {  …  }

 public ArrayList<T> getCheckedItems()
{…}

 @Override
 public int getCount() {return
mList.size(); }

 @Override
 public Object getItem(int position)
{return mList.get(position); }

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

```
@Override
  public View getView(int position, View
convertView,
ViewGroup parent) {  …  }

mCheckBox.setOnCheckedChangeListener(new
OnCheckedChangeListener() {

  @Override
   public void
onCheckedChanged(CompoundButton
buttonView,
     boolean isChecked) {
  …
 }

}
```

Implementation Using BaseAdapter 2-2

- Using the code, an application for demonstrating Adapters is created as shown in the following figure:

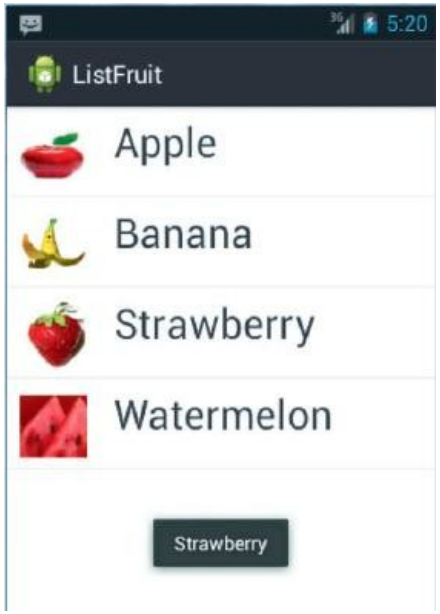


Advanced UI Components

- UI components act as an interface between the user and the application
- Advanced Android UI components:
  - Listview
  - ScrollView
  - TabBar
  - WebView
  - ViewFlipper
  - VideoView

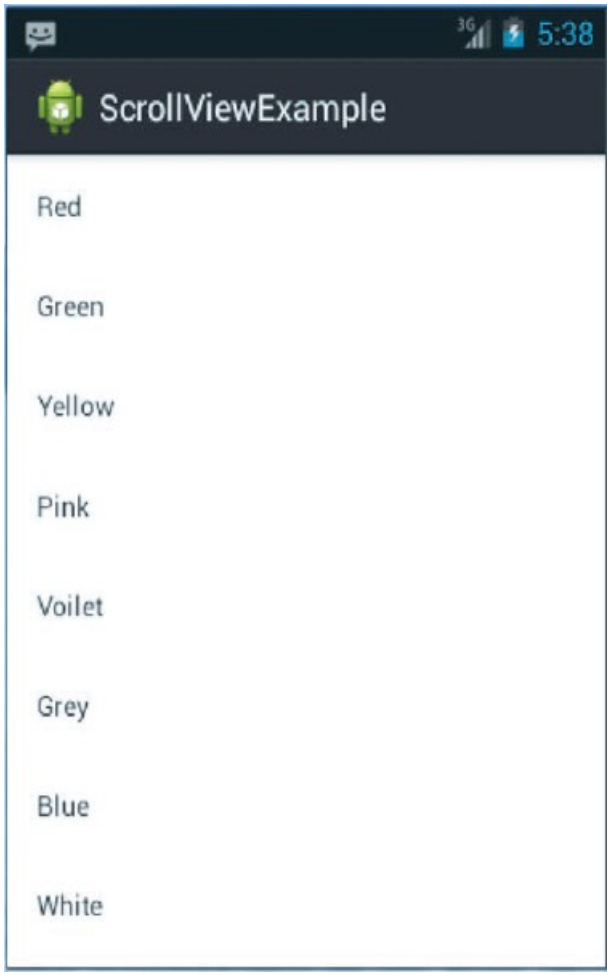
ListView

- ListView is used for displaying a list of scrollable data
- Data is displayed by using an object of the Adapter class
- The Adapter class acts as a bridge between view and the data source
- The Adapter is used for inserting data into the list that is retrieved from an array or a database using a query



ScrollView

- ScrollView is a container that holds child views
- It has a property of scrolling when the child items size in the container exceeds the screen size
- A ScrollView is a FrameLayout, meaning that the developer should place one child in it containing the entire contents to scroll
- ScrollView should not be used with ListView

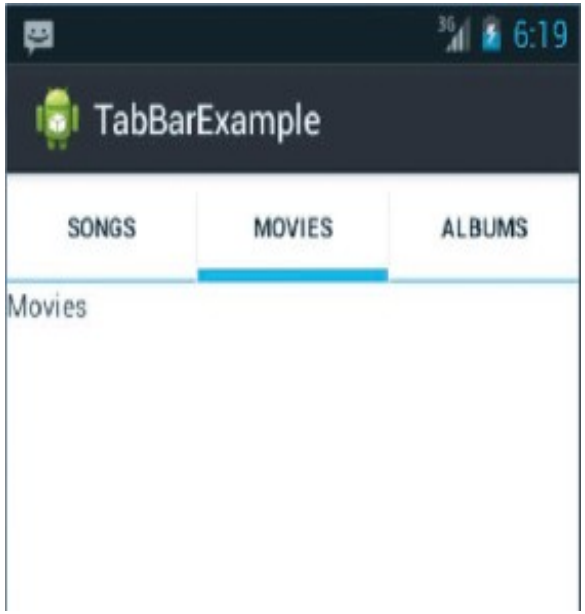


Following Code Snippet demonstrates an example for creating a ScrollView:

```
<ScrollView
android:layout_width="fill_parent"
android:layout_height="fill_parent" >

        <LinearLayout … >
                …
        </LinearLayout>
</ScrollView>
```

TabBar

- TabBar is used to display data in Tab format as is present in Android's contacts view
- It helps the developer to develop application for users where browsing for categorized data sets are required
- It can be scrollable, fixed, or stacked tabs
- TabHost is a container for a tabbed window view
- This object holds two children, a set of tab labels that the user clicks to select a specific tab and a FrameLayout object that displays the contents of that page



TabBar Example

Following Code Snippet demonstrates an example for creating a TabBar:

```
<TabHost
android:layout_width="match_parent" android:layout_height="match_parent" >
<LinearLayout">
<TabWidget
     android:id="@android:id/tabs"
     android:layout_width="match_parent"
     android:layout_height="wrap_content" >

                …

</TabWidget>

         <FrameLayout >

         …
         </FrameLayout>

</LinearLayout>
</TabHost>
```

WebView

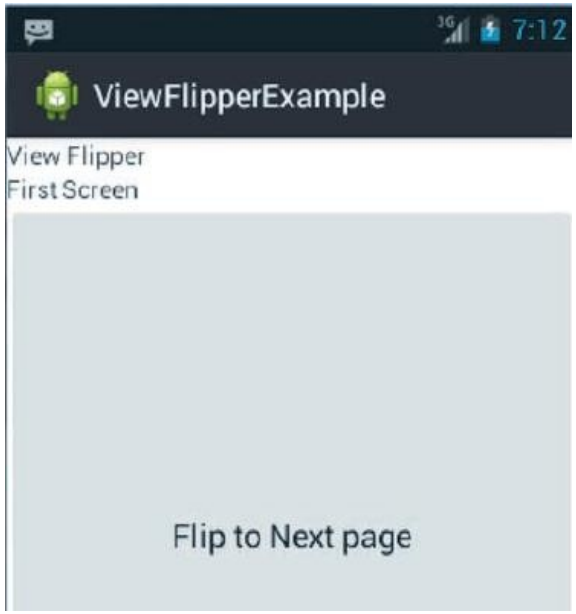
- WebView is used to display Web pages as a part of the activity layout
- It does not include any feature of Web browser
- WebView loads Web pages and renders raw HTML data

Following Code Snippet demonstrates an example for creating a WebView:

```
<?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" >

<WebView
android:id="@+id/webview01"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_weight="1" >
</WebView>
</LinearLayout>
```

ViewFlipper

- The ViewFlipper can be used to animate between two or more added views
- Here, only one child-view can be visible at a time
- User can navigate between child views. In ViewFlipper, flip-in and flip-out animations are used while navigating between child views
- The developer can also set automatic flipping between each child views at regular interval



Following Code Snippet demonstrates an example for creating a ViewFlipper:

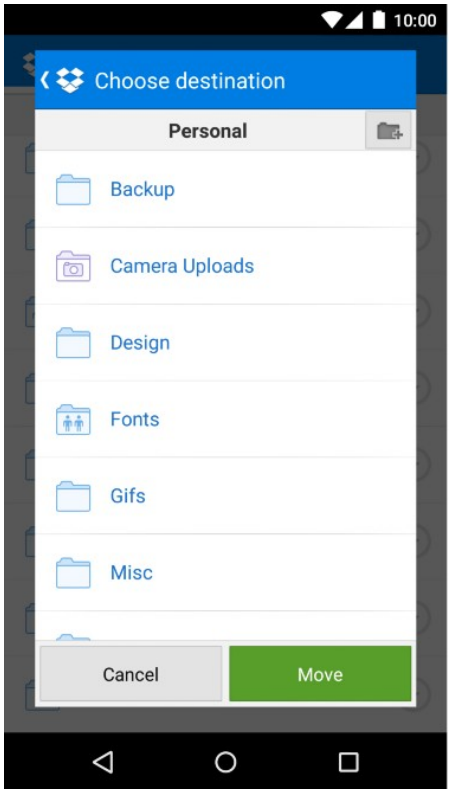
```
<LinearLayout … >
<ViewFlipper android:id="@+id/viewflipper" android:layout_width="fill_parent"
android:layout_height="fill_parent" >
 <LinearLayout >
 …
</LinearLayout>
</ViewFlipper>
</LinearLayout>
```

VideoView

- VideoView is used to play a video file
- Android 4.2 can support large extent of video formats including .aac format
- Lower versions cannot play all the video files of different formats

Dialogs

- A dialog is displayed in front of an activity as a small window
- When dialogs are active, the activity loses the focus
- A dialog can be created by extending the Dialog class
- It should be done by using any one of the following subclasses:
  - DatePicker Dialog
  - TimePicker Dialog
  - Progress Dialog
  - Alert Dialog
  - Toasts
  - Custom Dialog



DatePicker/TimePicker Dialog Example

- Following Code Snippet demonstrates an example for creating a DatePicker/TimePicker dialog:

```
public class TimePickerFragment extends DialogFragment implements
TimePickerDialog.OnTimeSetListener{
 private TimePickedListener mListener;
 @Override
 public Dialog onCreateDialog(Bundle savedInstanceState)
 {
        …
 }
 @Override
   public void onAttach(Activity activity)
 {
        …
 }
 public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        …
   }
   public static interface TimePickedListener{
    public void onTimePicked(Calendar time);
   }
 }
```

DatePicker/TimePicker Listener Example

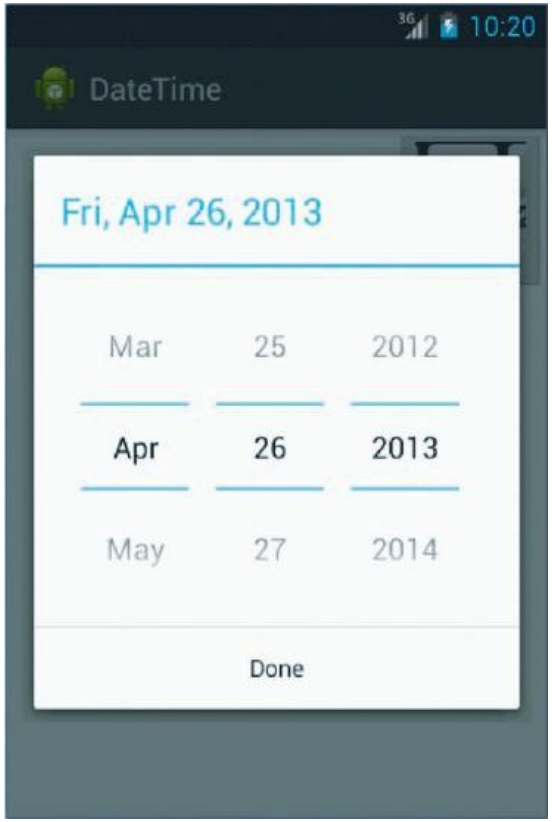
- Following Code Snippet demonstrates an example for creating a DatePicker/TimePicker listener:

```
public class Timepicker extends FragmentActivity implements TimePickedListener
{
@Override
 public void onCreate(Bundle savedInstanceState) {
  …

  mPickTimeButton.setOnClickListener(new OnClickListener() {
   public void onClick(View v) {

                DialogFragment newFragment =  new TimePickerFragment();
                new Fragment.show(getSupportFragmentManager(), "timePicker");
                }
 });
 }
        public void onTimePicked(Calendar time) {
                …
        }
}
```

DatePicker/TimePicker Application 1-3

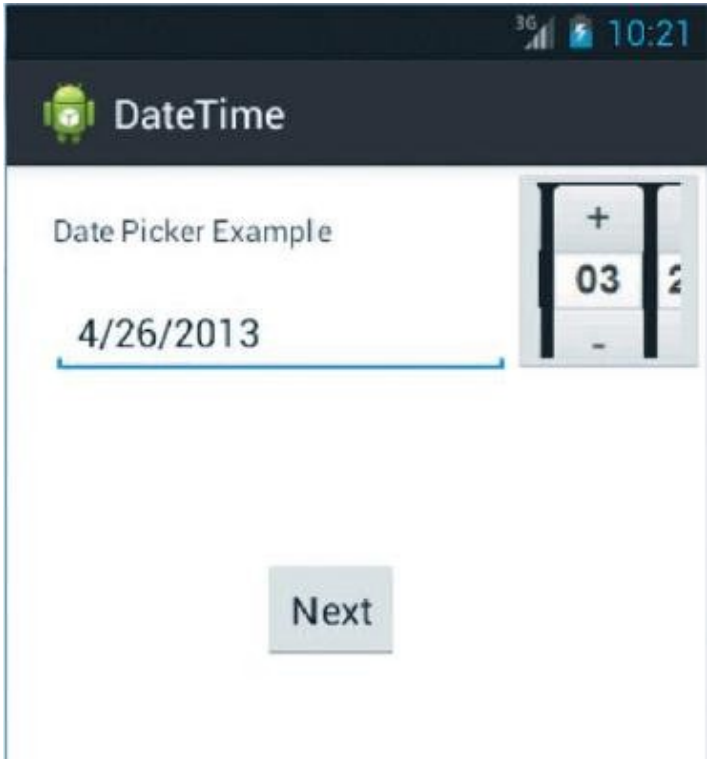
- Using the code, an application for demonstrating DatePicker/TimePicker Dialog is created as shown in the following figure:

- Once the date icon is clicked, the output will be as shown in the following figure:

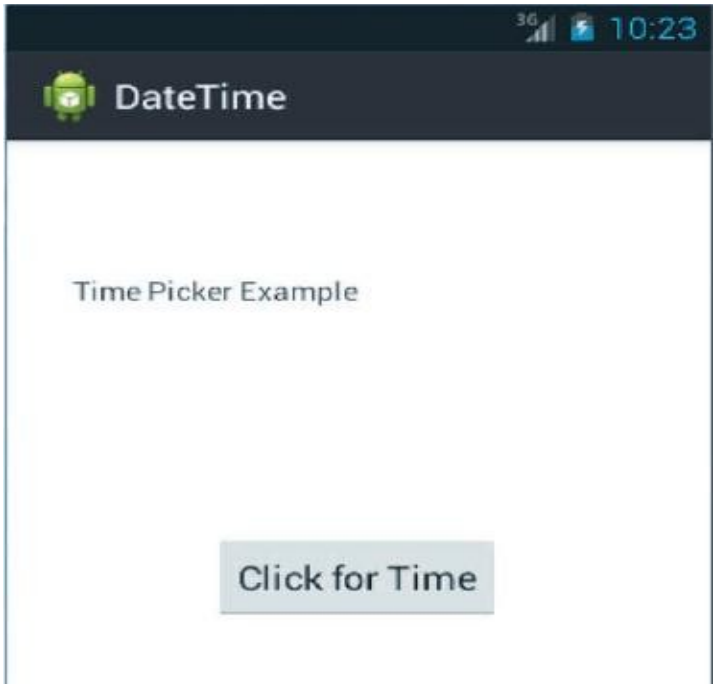


DatePicker/TimePicker Application 2-3

- Click Done and the output will be as shown in the following figure:

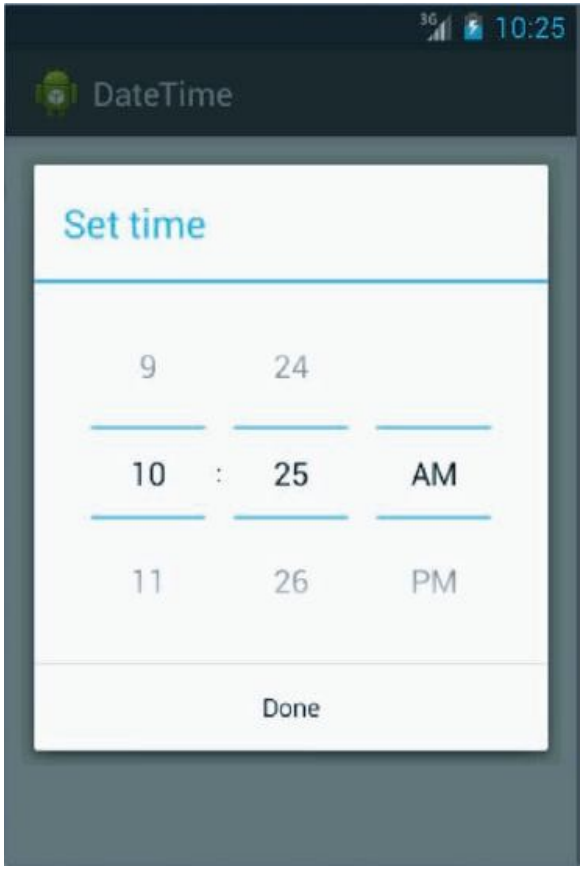


- Click Next and the output will be as shown in the following figure:



DatePicker/TimePicker Application 3-3

- Click the button ‘Click for Time’ and the output will be as shown in the following figure:



ProgressDialog Example 1-2

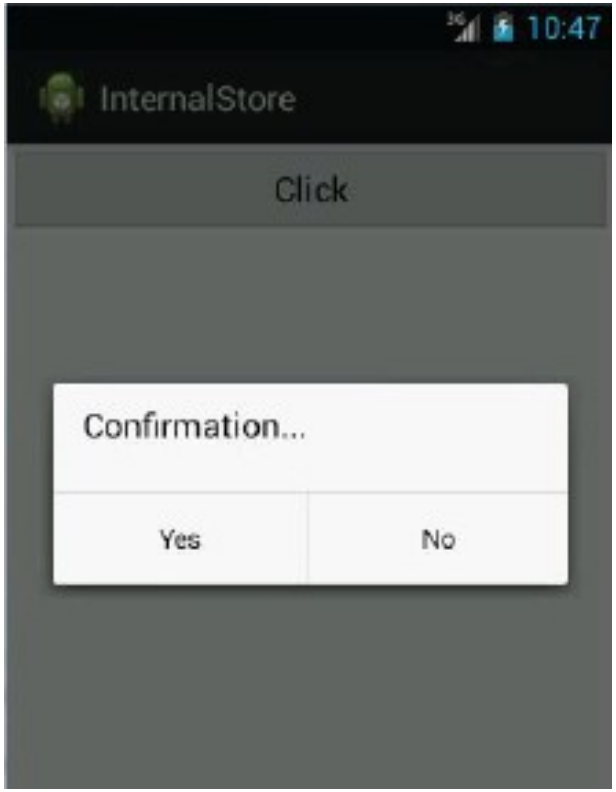
- Following Code Snippet demonstrates an example for creating a ProgressDialog:

```
final ProgressDialog myPd_ring = ProgressDialog.show(MainActivity.this,
"Please wait","Loading please wait..", true);
    myPd_ring.setCancelable(true);
…

    myPd_bar = new ProgressDialog(MainActivity.this);
    myPd_bar.setMessage("Loading....");
    myPd_bar.setTitle("Please Wait..");
    myPd_bar.setProgressStyle(myPd_bar.STYLE_HORIZONTAL);
    myPd_bar.setProgress(0);
    myPd_bar.setMax(30);
    myPd_bar.show();

…

myPd_bar.incrementProgressBy(5);
```

ProgressDialog Example 2-2

- Using the explained code, an application for demonstrating ProgressDialog is created as shown in the following figure:

- Clicking the button will display the output as shown in the following figure:



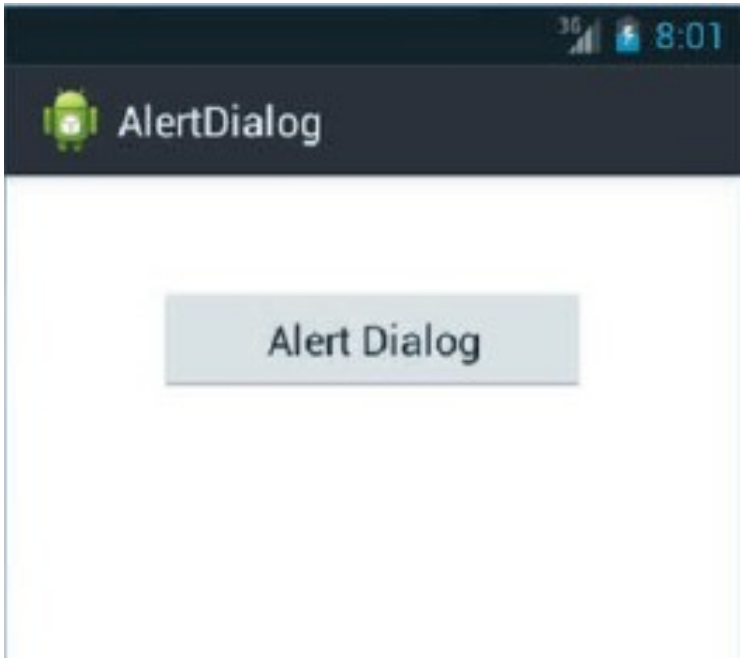
AlertDialog Example 1-2

- Following Code Snippet demonstrates an example for creating a AlertDialog:

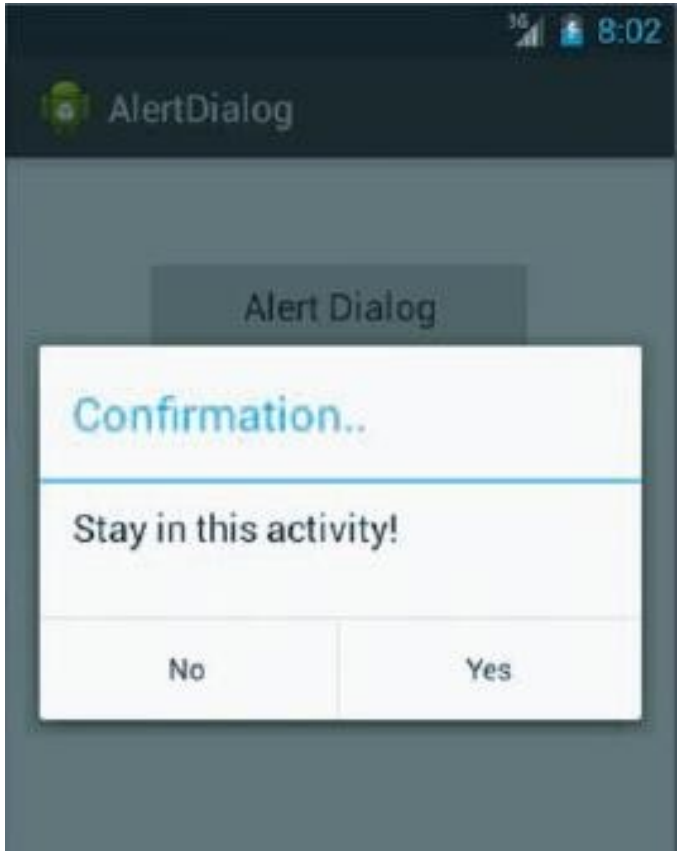
```
 AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(context);
alertDialogBuilder.setTitle("Confirmation..");
alertDialogBuilder.setMessage("Stay in this activity!")
      .setCancelable(false)
.setPositiveButton("Yes",
         new DialogInterface.OnClickListener() {
         publicvoid onClick(DialogInterface dialog,int id) {
          dialog.cancel();
         }
        })
.setNegativeButton("No", new DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog,int id) {
                  …
                  MainActivity.this.finish();
         }
        });
AlertDialog alertDialog = alertDialogBuilder.create();
alertDialog.show();
```

AlertDialog Example 2-2

- Using the explained code, an application for demonstrating AlertDialog is created as shown in the following figure:



- Clicking the button will display the output as shown in the following figure:



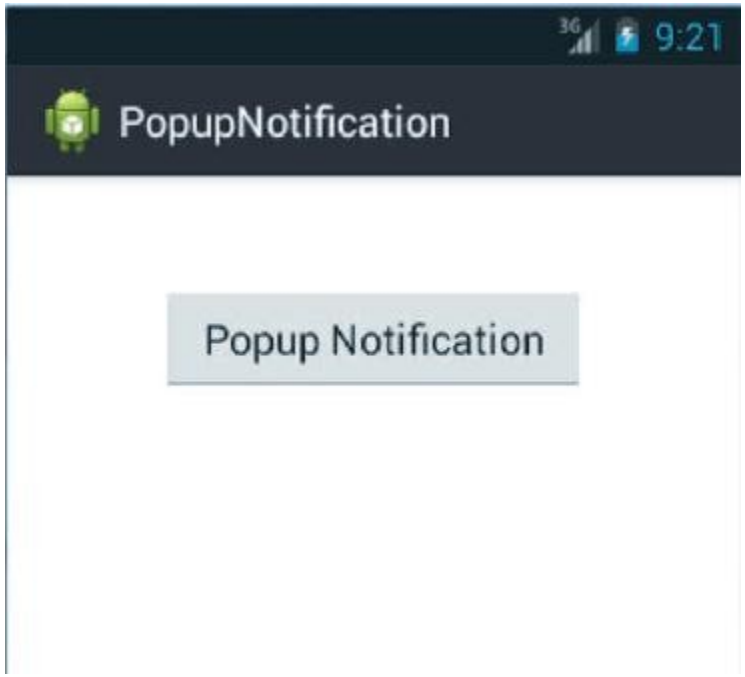
PopupDialog Example 1-2

- Following Code Snippet demonstrates an example for creating a PopupDialog:

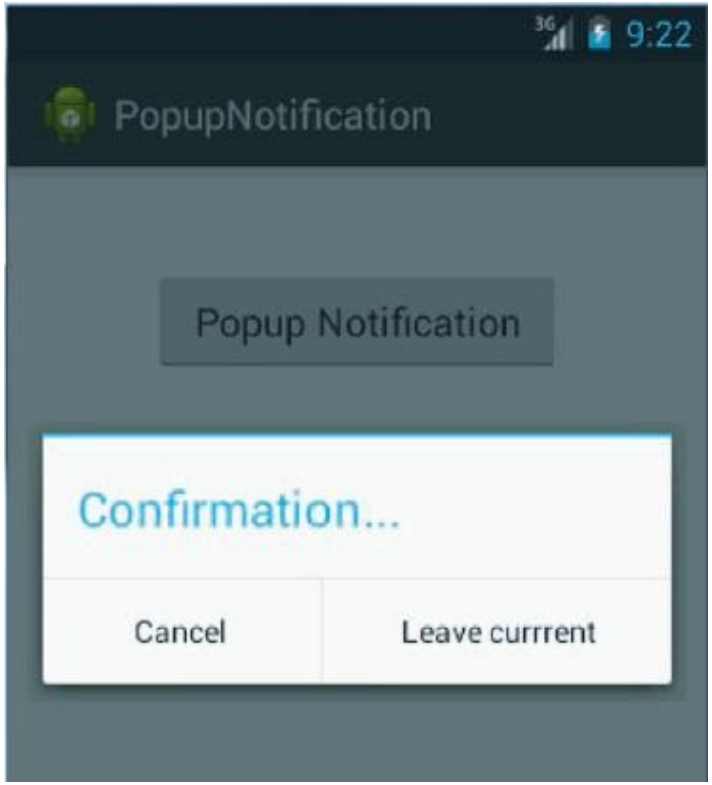
```
AlertDialog.Builder alert_dialog = new
AlertDialog.Builder(context);
    alert_dialog.setTitle("Confirmation...");
    alert_dialog.setNegativeButton("Cancel",
      new DialogInterface.OnClickListener() {
      @Override
         publicvoid onClick(DialogInterface dialog,int which) {
       }
      });
    alert_dialog.setPositiveButton("Leave current",
      new DialogInterface.OnClickListener() {
      @Override
         public void onClick(DialogInterface dialog, int which) {
        finish();
       }
      });
    alert_dialog.show();
```

PopupDialog Example 2-2

- Using the explained code, an application for demonstrating PopupDialog is created as shown in the following figure:



- Clicking the button will display the output as shown in the following figure:



Toast Example

- Following Code Snippet demonstrates an example for creating a Toast:

```
Toast.makeText(getApplicationContext(),
      "This is a Toast Notification", Toast.LENGTH_LONG).show();
```

- Using the code, an application for demonstrating Toast is created as shown in the following figure:

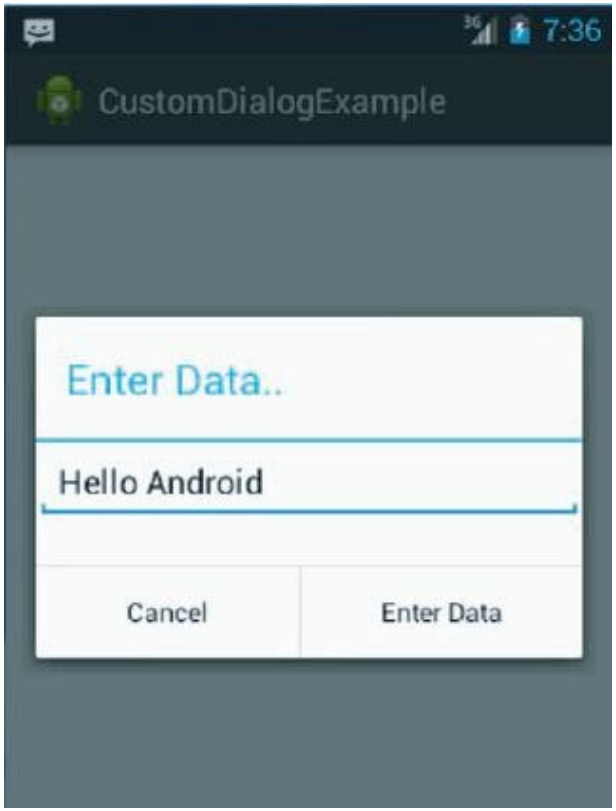
CustomDialog Example 1-2

- Following Code Snippet demonstrates an example for creating a CustomDialog with an EditText and an input check:

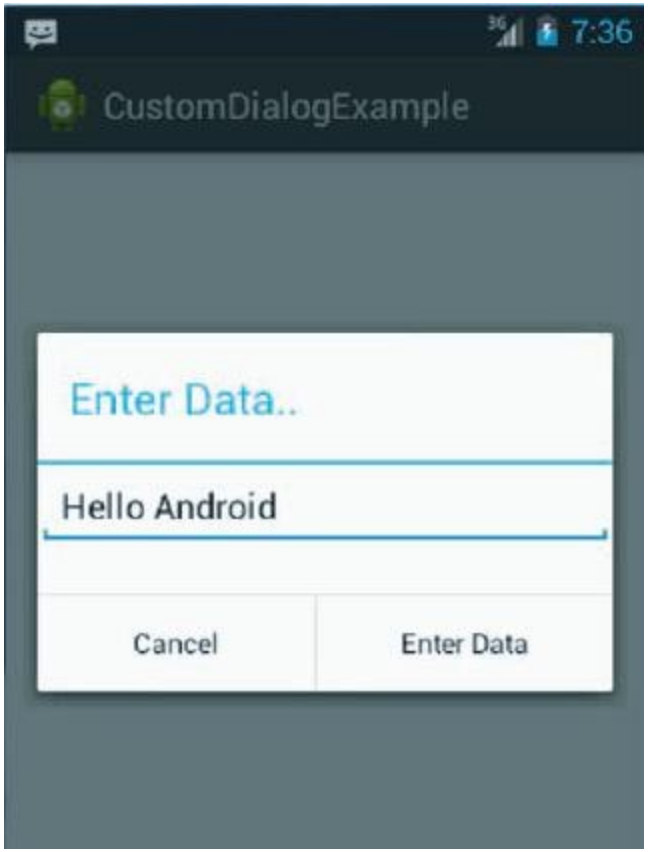
```
AlertDialog.Builder alert = new AlertDialog. Builder(MainActivity.this);
userData = new EditText(MainActivity.this);
alert.setTitle("Enter Data..");
alert.setView(userData);
alert.setPositiveButton("Enter Data", new OnClickListener() {
          @Override
          public void onClick(DialogInterface dialog, int which) {
                    if (!userData.getText().toString(). equals("")) {
           Toast.makeText(getApplicationContext(),"Data Entered is : " +
userData.getText().toString(), Toast.LENGTH_LONG).show();
                    } else {
                    Toast.makeText(getApplicationContext(),
"Field should not be empty", Toast.LENGTH_LONG).show();
                    }
          }
});
alert.setNegativeButton("Cancel", new OnClickListener() {
          @Override
          public void onClick(DialogInterface dialog, int which) {
                              }
 });
alert.show();
```

CustomDialog Example 2-2

- Using the explained code, an application for demonstrating CustomDialog is created as shown in the following figure:



- When the user clicks Enter Data in the dialog, the output will be as shown in the following figure:



Widgets

- App widgets are miniature application views that can be embedded in other applications and receive periodic updates
- Application widgets are useful to provide control to the application without using the entire screen
- Android comes with several pre-installed widgets such as the Analog clock, Settings, and so on
- Widgets can be displayed on the home screen on any of the screens selected by the user
- The developer has no control over where the widget is displayed or the position of the widget
- Lock Screen widget support was introduced in API level 17 (Android 4.2) and dropped in API level 22 (Android 5.1)

Widget Manifest and Metadata Example

- Following Code Snippet demonstrates the manifest file changes that are required to be made for widgets:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
   ...
   android:widgetCategory="keyguard|home_screen">
</appwidget-provider>

<meta-data android:name="android.appwidget.provider"
    android:resource="@xml/quote_appwidget_info" />
```

- Following Code Snippet demonstrates the meta data file contents that are required to be made for widgets:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="86400000"
    android:previewImage="@drawable/preview"
    android:initialLayout="@layout/quote_appwidget"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen|keyguard"
    android:initialKeyguardLayout="@layout/quote_keygaurd">
</appwidget-provider>
```

Determining Display Location/Creating Custom Widget

- Following Code Snippet demonstrates the process for detecting where the widget is being displayed:

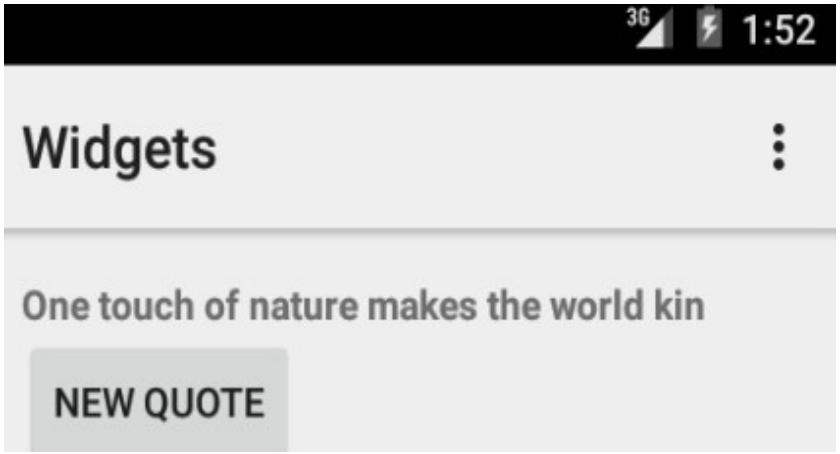
```
Bundle myOptions = appWidgetManager.getAppWidgetOptions (appWidgetId);
int category = myOptions.getInt(AppWidgetManager.OPTION_APPWIDGET_HOST_CATEGORY, -1);
boolean isKeyguard = (category == AppWidgetProviderInfo.WIDGET_CATEGORY_KEYGUARD);
int widgetLayout = isKeyguard ? R.layout.quote_keygaurd : R.layout.quote_appwidget;
```

- Following Code Snippet demonstrates the code for creating an Application Widget by extending the AppWidgetProvider class:

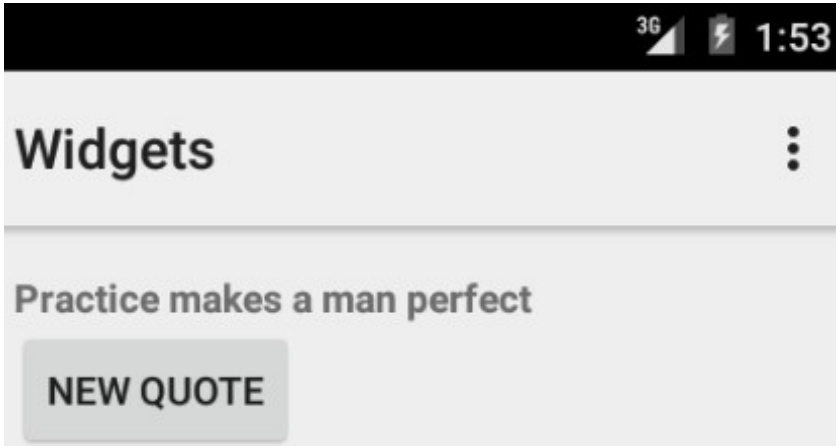
```
public class QuoteAppWidgetProvider extends AppWidgetProvider {
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[]
appWidgetIds) {
        …
        }
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        super.onReceive(context, intent);
…
        }
    }
}
```

Application Widgets Example 1-4

- Using the explained code, an application for demonstrating Widgets is created as shown in the following figure:

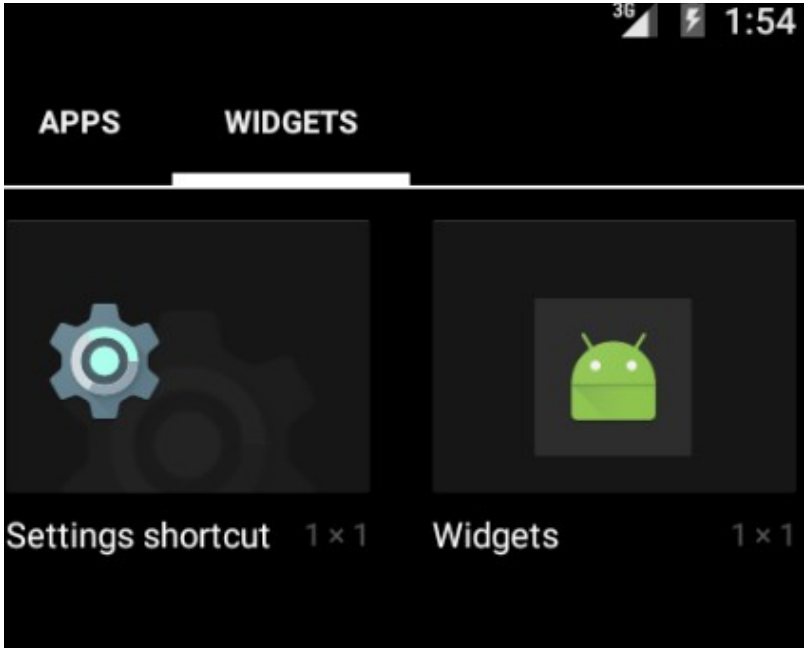


- Clicking on New Quote will show a new quote as shown in the following figure:



Application Widgets Example 2-4

- The user can exit the application and return to the app drawer to add the new widget as shown in the following figure:

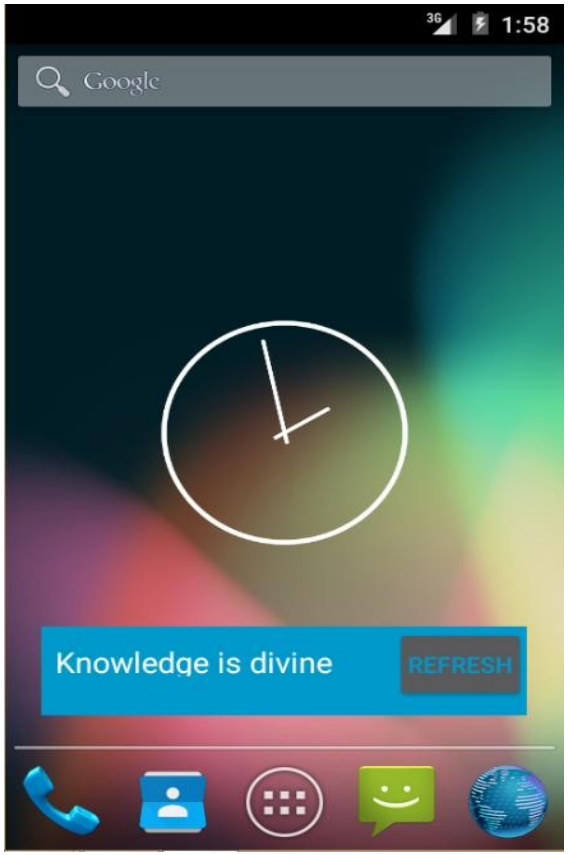


- The widget can be added to the home screen as shown in the following figure:

Application Widgets Example 3-4

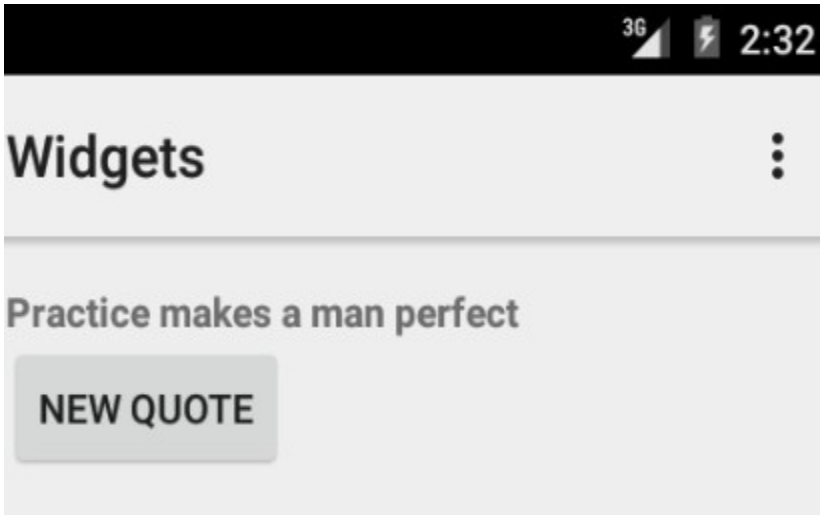
- The widget is displayed on the home screen as shown in the following figure:

- By clicking the Refresh button a new quote is displayed as shown in the following figure:



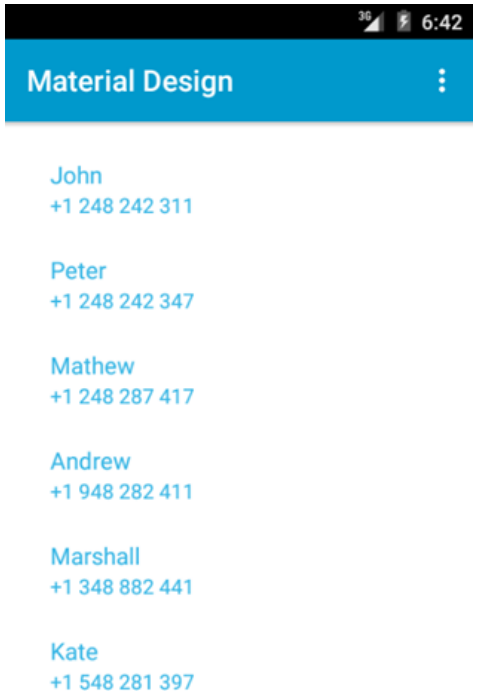
Application Widgets Example 4-4

- By clicking the Quote itself, the application will be loaded as shown in the following figure:



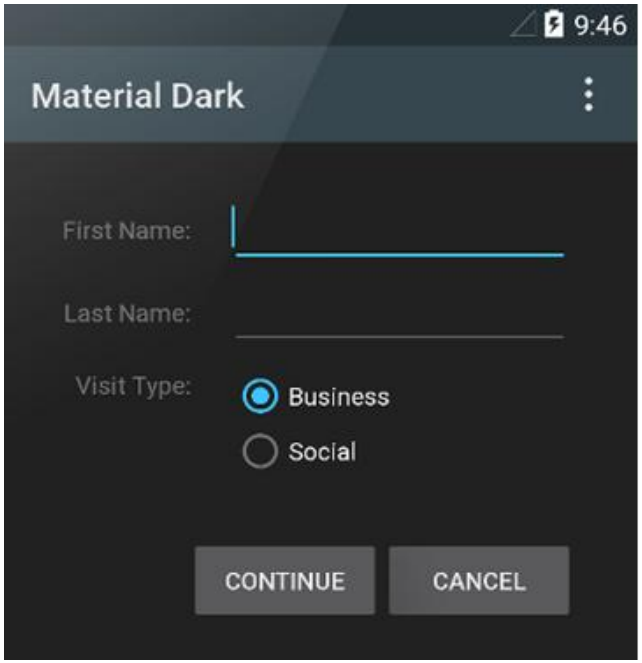
Material Design Philosophy

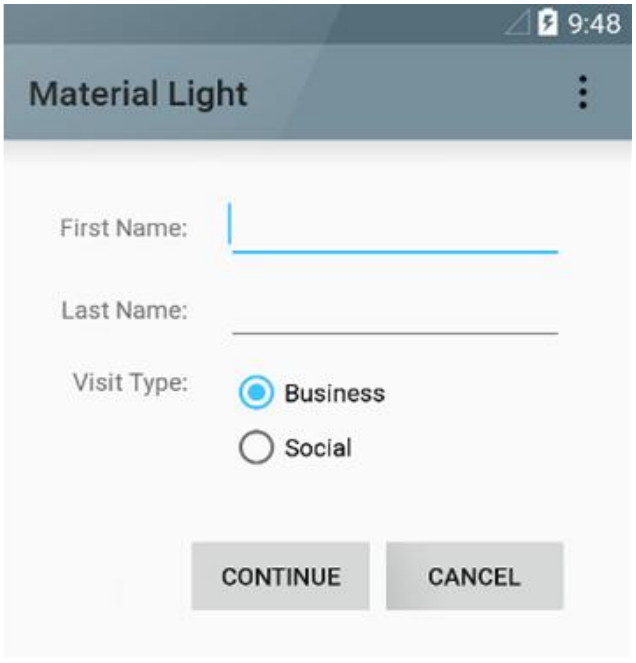
- With the release of Android Lollipop, Google has introduced a new UI design philosophy called **Material Design**
- The material design specifies a set of design guidelines to maintain a look, consistent with the base operating system
- It is not mandatory to follow the Material Design guidelines
- Google strongly recommends doing so, at least at a base level



Material Theme

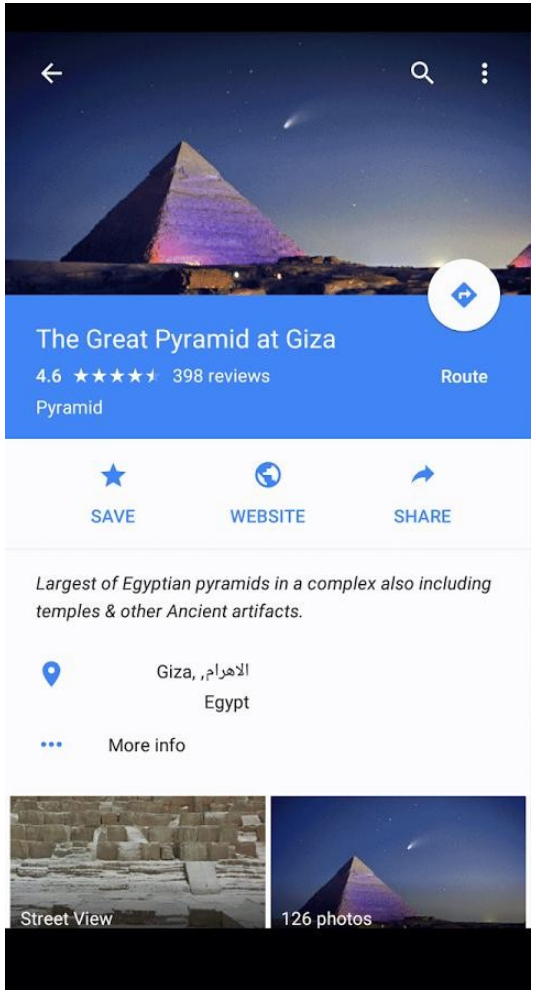
- The simplest guideline towards a material look and feel is utilizing the system themes provided by Google
- API level 20 (Android 5.0) comes with Theme.Material (Dark Version) and Theme.Material.Light (Light Version) themes
- The developer can set the theme by navigating to res → values → styles.xml





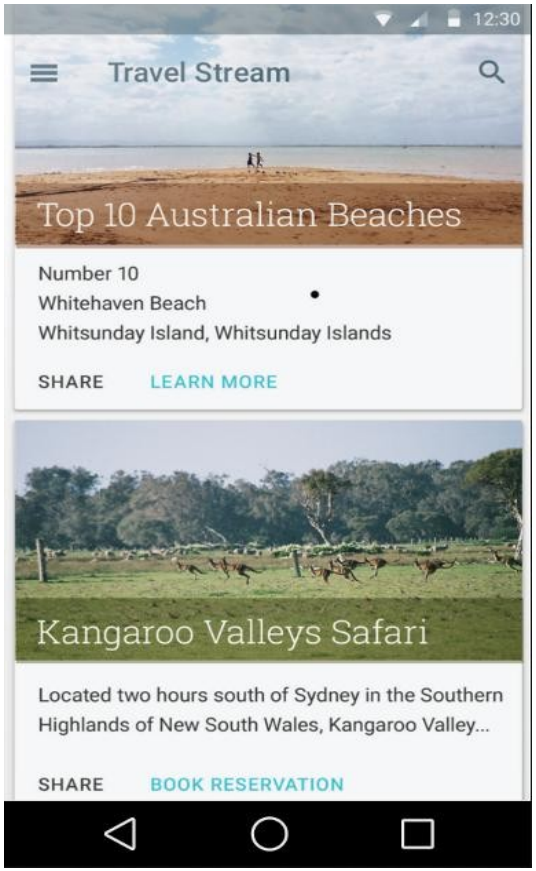
Transparent Activity Bars and Full Screen

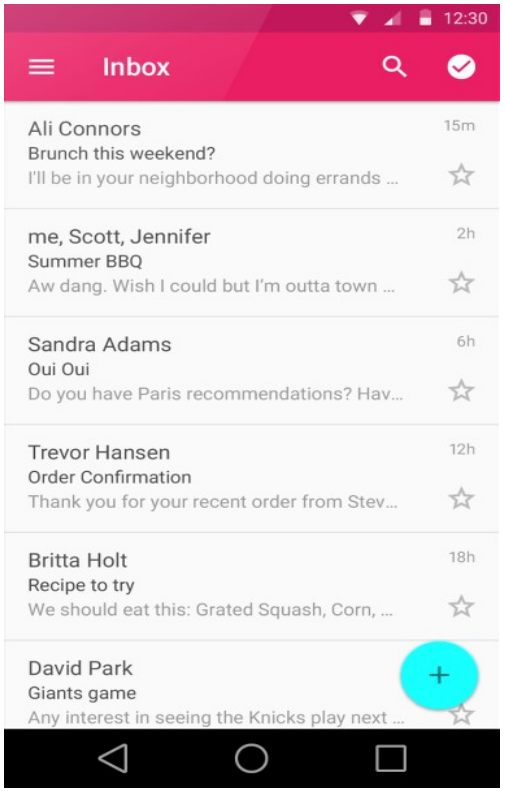
- Android Lollipop brings the ability to use transparent Activity bars and to use the entire screen without the notification and the navigation bar
- The material design philosophy recommends utilizing this feature when the activity content is static and prominently multimedia information



Lists and Cards

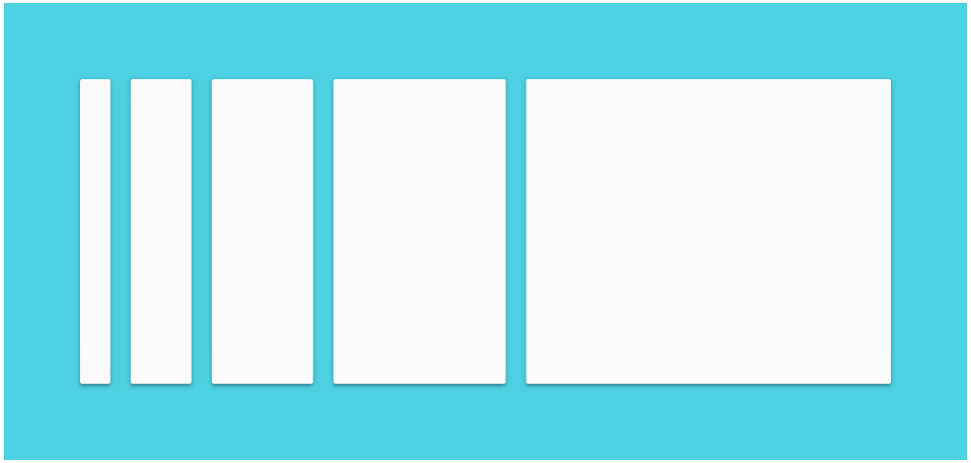
- Google recommends the developers to use the RecyclerView instead of the standard ListView
- The RecyclerView comes with a more flexible layout and fully integrates with the Material Design philosophy
- The API to use a RecyclerView is similar to that of a ListView
- The official Gmail app utilizes the ReyclerView





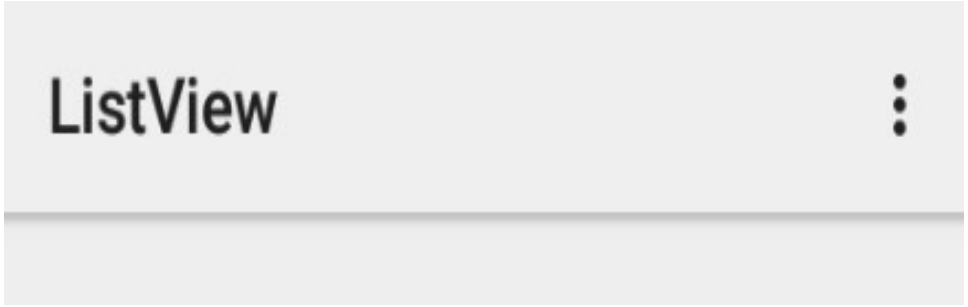
Uniform Layouts

- Google recommends designing the application layout as multiples of a unit measurement
- Dividers of uniform size are recommended whenever a side panel or a navigational menu needs to be displayed
- This is contradictory to the philosophy of the metro theme of windows, where each ‘tile’ is deliberately made of different dimensions



Shadows

- Shadows and elevations are recommended to be used wherever applicable
- The android:elevation property can be used to set the elevation value of a view
- The property makes the view to appear elevated or hovering above the rest of the layout



Animations

- The material design specification recommends using animations to transition between activities appear more natural
- A fade out animation can be used if a fresh unrelated activity is being loaded. Touch feedback is recommended on clickable objects
- Circular reveals are recommended to be used whenever the Activity being displayed is changing the screen orientation

Color Schemes

- Material design utilizes Bright color schemes throughout the application
- The developer is strongly recommended to style the app as per the brand requirement
- The ability to color the Action and the notification bar allows for full customization of look and feel
- The developer can use the holo car pallete using '@android:color/holo_'
- The color palette can be used to determine the right theme to go along with the application interface

Maintaining Compatibility

- Earlier versions of Android do not fully support certain feature essential to the material
- In order to deal with the issues with Material widgets, the developer can use the v7 App Compat Support library
- Alternative styles can be designed for older versions by separating the styles.xml for older versions and styles-v21.xml for devices supporting API level 21 and higher
- The color scheme and the layout schemes are compatible with any version of Android
- The Material theme may not be available to inherit from in this scenario. As a workaround, the developer can rewrite the entire theme, to ensure the consistency of the look of the application

Summary

- Adapter is an object that acts as a bridge between the UI Components such as List view, Grid view, and so on and the underlying data sources
- User Interface components act as an interface between the user and the application to input data and display the expected result
- A dialog is displayed in front of an activity as a small window and can be a DatePickerDialog, TimePickerDialog, ProgressDialog, or AlertDialogs
- Popups are smaller versions of dialogs which are designed in such a way so that users can make a selection to move forward or click outside the popup
- Toast is a popup that displays feedback for an activity
- Custom Widgets can be created by extending the AppWidgetProvider class. They can be placed on the home screen or the lock screen
- Lock screen widget support was dropped in API level 22
- Material Design is Google's new optional design philosophy for creating application UI