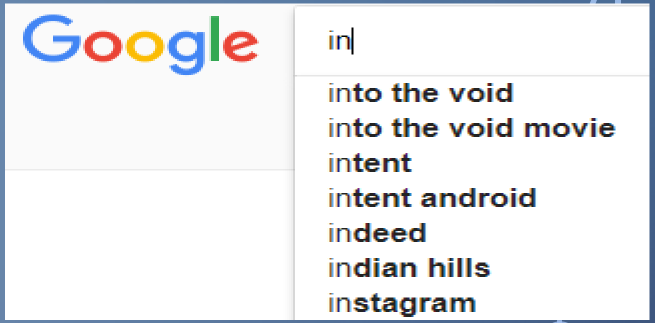
LESSON – 5
ADVANCED UI COMPONENTS

AGENDA

- AutoCompleteTextView(ACTV)
- Spinner
- ListView
- Adapters – ArrayAdapter(API)
- Adapters – Custom Adapter(User Defined)
- Menus & ActionBar
- Dialogs
- Notifications

DAY - 1

AutoCompleteTextView



- The AutoCompleteTextView is sort of a hybrid between the EditText (field) and the Spinner. With auto-completion, as the user types, the text is treated as a prefix filter, comparing the entered text as a prefix against a list of candidates.
- Matches are shown in a selection list that folds down from the field. The user can either type out an entry (e.g., something not in the list) or choose an entry from the list to be the value of the field.
- In addition, AutoCompleteTextView has an Threshold property, to indicate the minimum number of characters a user must enter before the list filtering begins.
- To create a UI Component use <AutoCompleteTextView> tag in XML.
- To provide an auto completion support to the user first we have to configure the values. Configure the values either using
  - XML approach or Java approach.

XML APPROACH

Step 1: In your Android Project go to the folder app→res→values→strings.xml

Set the values by using the given lines of codes

```
<string-array name="array_name">
        <item>value1</item>
        <item>value2</item>
        .............
</string-array>
```

Step 2: Create an AutoCompleteTextView UI component in XML with an id.

Step 3: Use the following code to get the Xml configured values into your Activity.

```
String[] values=getResources().getStringArray(R.array.array_name);
```

Step 4: for presenting the values we have to create an Adapter, in Android there are 3 types of Adapters.

a) ArrayAdapter b) Custom Adapter(will discuss in ListView) c) Cursor Adapter(will discuss in SQLite)

```
ArrayAdapter<String> adapter= new ArrayAdapter<String>(context,xml_file,values);
actv.setAdapter(adapter); // To specify the Adapter Object. Here actv is an Autocomplete TextView type
actv.setThreshold(int); // To specify the dropdown support, after how many characters of user entry
```

Hands-on-example – Countries list using ACTV

strings.xml

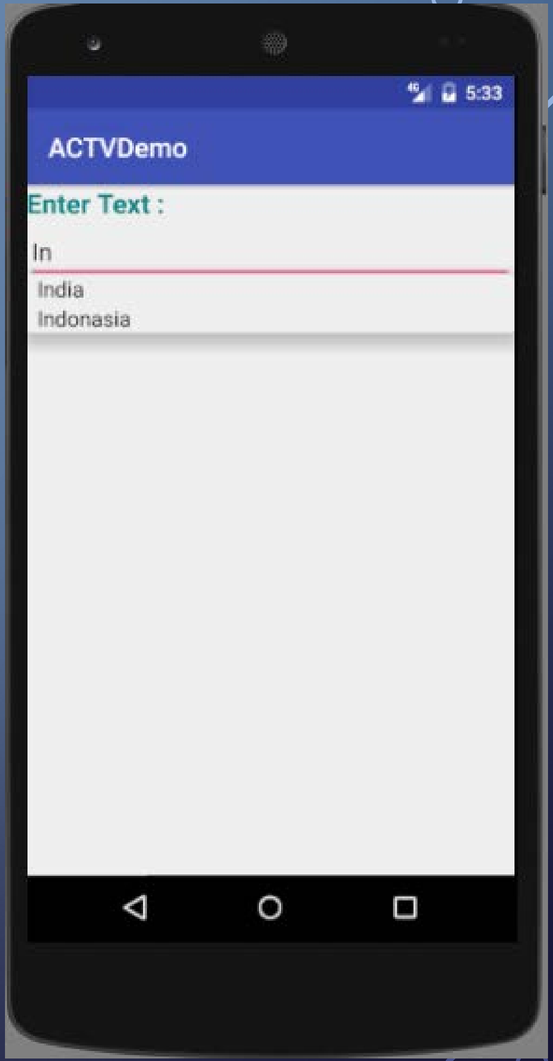
```
<resources>
    <string name="app_name">ACTVDemo</string>
    <string-array name="countries">
        <item>India</item>
        <item>Indonasia</item>
        <item>USA</item>
        <item>Asia</item>
        <item>Africa</item>
        <item>Siria</item>
        <item>Sri Lanka</item>
        <item>Canada</item>
        <item>Koria</item>
        <item>Island</item>
        <item>Bangaladesh</item>
        <item>Nepal</item>
    </string-array>
</resources>
```

XML Code

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Enter Text : "
        android:textSize="20sp"
        android:textStyle="bold"
        android:textColor="#008888"
        />
     <AutoCompleteTextView
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:id="@+id/actv"/>
</LinearLayout>
```

```
public class MainActivity extends AppCompatActivity {
  AutoCompleteTextView actv;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      actv = (AutoCompleteTextView) findViewById(R.id.actv);
      // Get the XML configured vales into the Activity and stored into an String Array
      String[] values = getResources().getStringArray(R.array.countries);
      /* Pass three parameters to the ArrayAdapter
       1. The current context,
       2. The resource ID for a built-in layout file containing a TextView to use when instantiating views,
          which are available in android.R.layout
       3. The objects to represent in the values
       */
      ArrayAdapter<String> adapter = new
ArrayAdapter<String>(this,android.R.layout.simple_spinner_item,values);
      actv.setAdapter(adapter);
      actv.setThreshold(2); // by default 1
  }
}
```

Output



ACTVDemo

Enter Text :

In

India

Indonasia

Java Approach & OnItemClickListener

```
public class MainActivity extends AppCompatActivity {
    AutoCompleteTextView actv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        actv = (AutoCompleteTextView) findViewById(R.id.actv);
String[] values = new String[]{"Asia","Australia","America","Belgium","Brazil","Canada","California","Dubai","France","Paris" };
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,android.R.layout.simple_spinner_item,values);
        actv.setAdapter(adapter);
        actv.setThreshold(2); // by default 1
        actv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                Toast.makeText(getApplicationContext(),"Item selected is : "+parent.getItemIdAtPosition(position),Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Toast is one of the notification method in Android which is used to display text on the screen for few seconds.

Toast

Toast makeText (Context context, CharSequence msg, int duration).show()

Parameters	
context	Context: The context to use. Usually your Application or Activity object.
msg	CharSequence: The text to show. Can be formatted text.
duration	int: How long to display the message. Either LENGTH_SHORT or LENGTH_LONG

AdapterView.OnItemClickListener

- Interface definition for a callback to be invoked when an item in this AdapterView has been clicked. Implementers can call getItemAtPosition(position) if they need to access the data associated with the selected item.

Public methods	
abstract void	onItemClick(AdapterView<?> parent, View view, int position, long id) Callback method to be invoked when an item in this AdapterView has been clicked.

Parameters	
parent	AdapterView: The AdapterView where the click happened.
view	View: The view within the AdapterView that was clicked (this will be a view provided by the adapter)
position	int: The position of the view in the adapter. (position starts from zero)
id	long: The row id of the item that was clicked.

Difference Between XML And Java Approach

XML Approach

- To provide static data
  - Example : Country names are static
- To provide Multi language support-I18N-(Internationalization) - (Discussed later in Localization)

- Java Approach
- To provide dynamic data
  - Example : Movies list
- Can not provide Multi language support.

Spinners

- Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.
- You can add a spinner to your layout with the Spinner object. You should usually do so in your XML layout with a <Spinner> element. For example:
-
```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/planets_array"
/>
```

Populate the spinner with user choices

- The choices you provide for the spinner can come from any source, but must be provided through an SpinnerAdapter, such as an ArrayAdapter if the choices are available in an array or a CursorAdapter if the choices are available from a database query.
- For instance, if the available choices for your spinner are pre-determined, you can provide them with a string array defined in a string resource file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

Responding to user selections

- When the user selects an item from the drop-down, the Spinner object receives an on-item-selected event.
- To define the selection event handler for a spinner, implement the AdapterView.OnItemSelectedListener interface and the corresponding onItemSelected() callback method. For example, here's an implementation of the interface in a
- public class SpinnerActivity extends Activity implements OnItemSelectedListener {

```
    ...
    public void onItemSelected(AdapterView<?> parent, View view,
            int pos, long id) {
        // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
    }
    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}
```

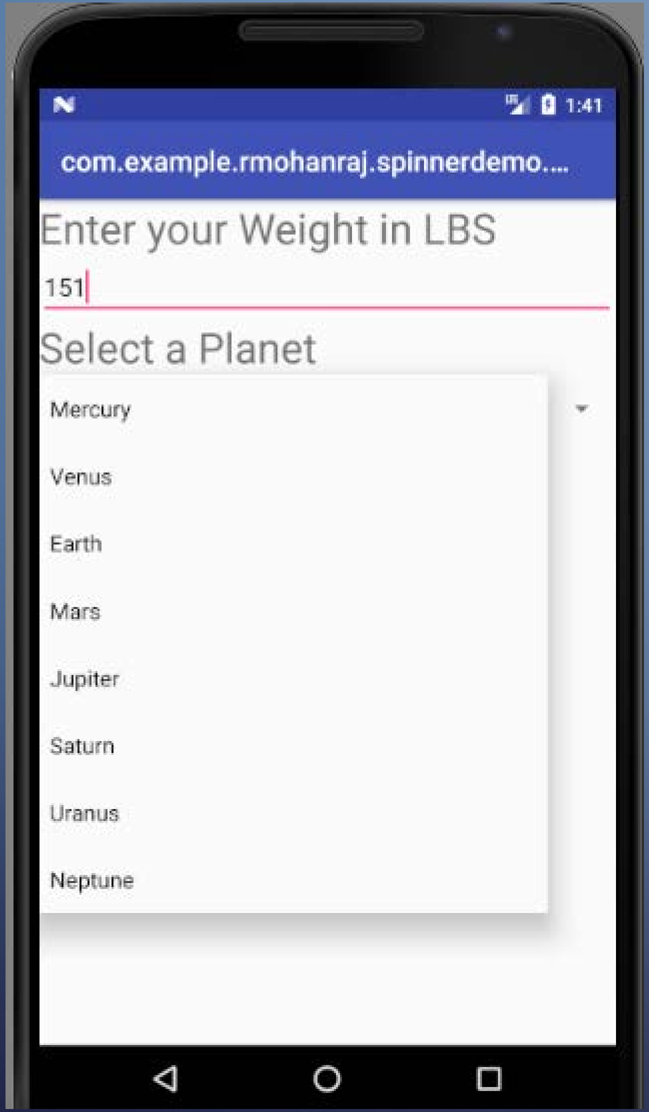
- The AdapterView.OnItemSelectedListener requires the onItemSelected() and onNothingSelected() callback methods.
- Then you need to specify the interface implementation by calling setOnItemSelectedListener():

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);

spinner.setOnItemSelectedListener(this);
```

Example from Demo : Lesson5\SpinnerDemo folder

Hands on Example for Spinner - Planet Weight Calculator

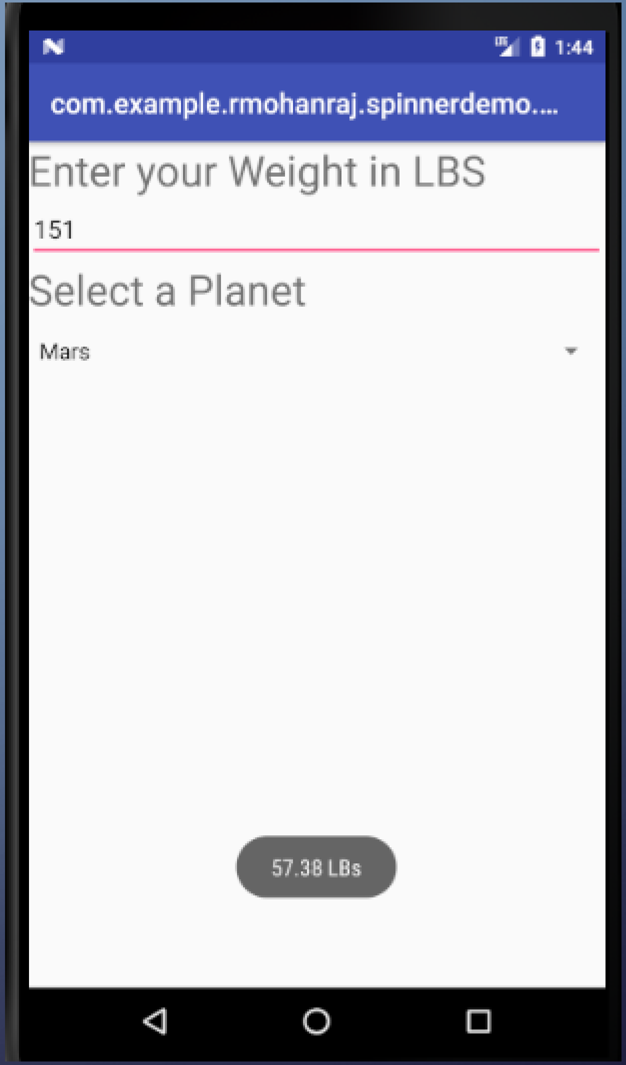

com.example.rmohanraj.spinnerdemo....

Enter your Weight in LBS

151

Select a Planet

- Mercury
- Venus
- Earth
- Mars
- Jupiter
- Saturn
- Uranus
- Neptune


com.example.rmohanraj.spinnerdemo....

Enter your Weight in LBS

151

Select a Planet

Mars

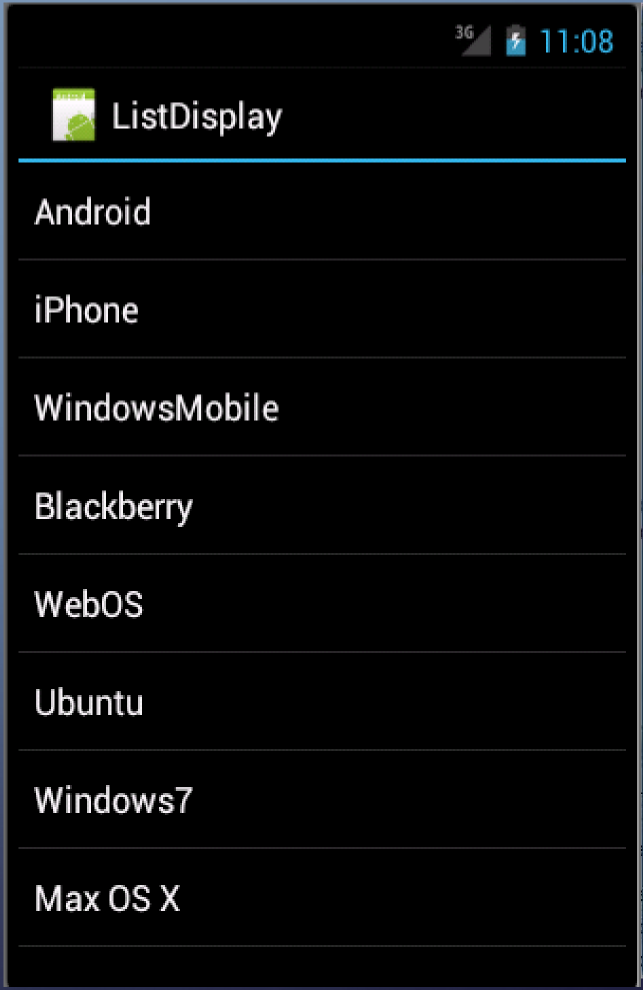
57.38 LBs

ListView

- ListView is a view group that displays a list of scrollable items. An Adapter that pulls content from a source, such as, a query or an array, helps to insert the list items automatically. Each item result is converted into a View and added to the list by the Adapter.

```
 <ListView   android:id="@android:id/list"
android:layout_width="match_parent"
android:layout_height="wrap_content"
</ListView>
```

Simple ListView



36 11:08

ListDisplay

Android

iPhone

WindowsMobile

Blackberry

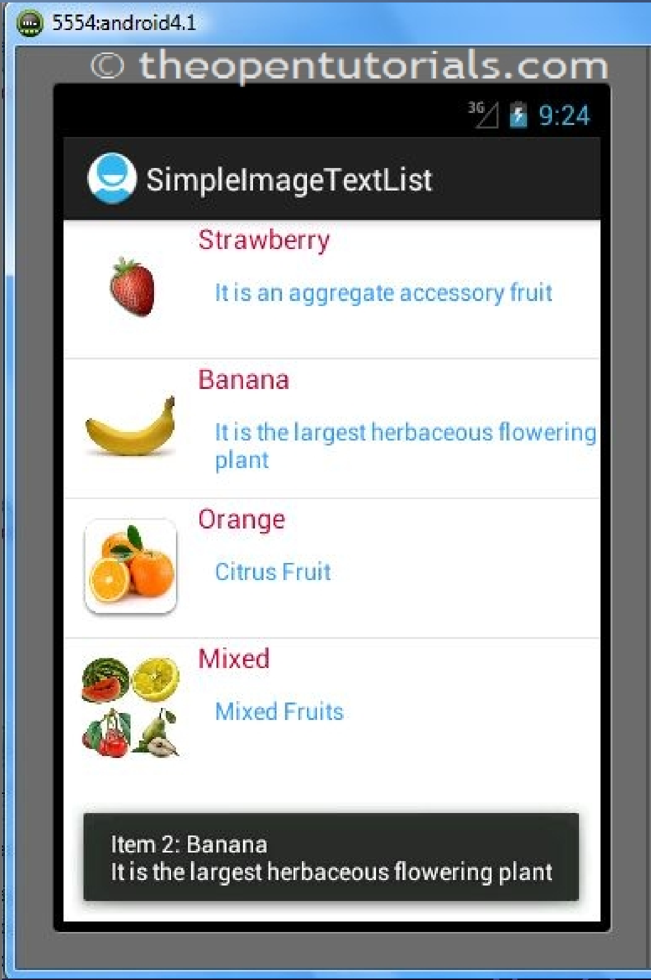
WebOS

Ubuntu

Windows7

Max OS X

Customized ListView



5554:android4.1

© theopentutorials.com

36 9:24

SimpleImageTextList

Strawberry

It is an aggregate accessory fruit

Banana

It is the largest herbaceous flowering plant

Orange

Citrus Fruit

Mixed

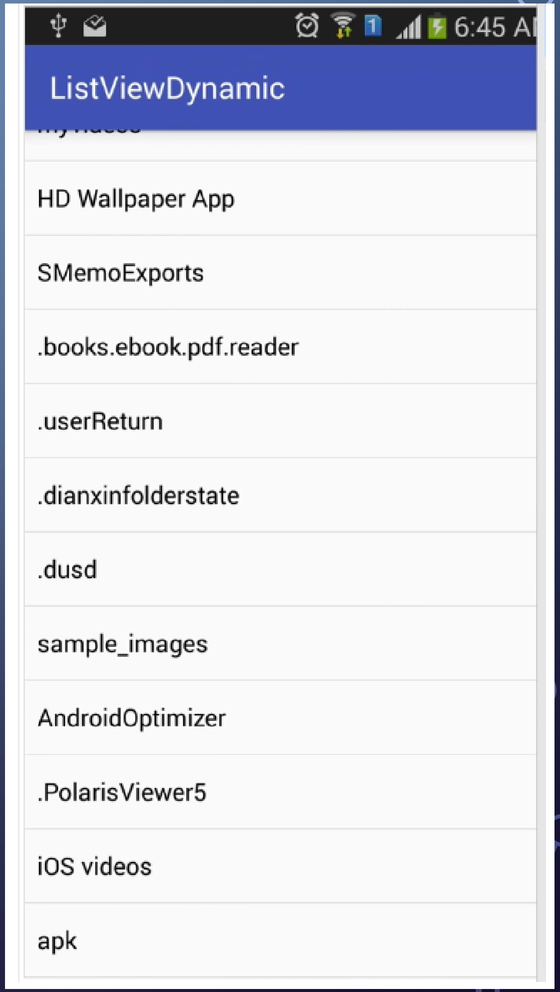
Mixed Fruits

Item 2: Banana
It is the largest herbaceous flowering plant

Simple ListView- Hands on Example

- **Problem :** To get the files from the Phone and View in a ListView. We are working with Internal Storage in this lesson. Remaining will discuss later.
- Android provides several storage options for you to save persistent application data.
  - Shared Preferences
    - Store private primitive data in key-value pairs.
  - Internal Storage
    - Store private data on the device memory.
  - External Storage
    - Store public data on the shared external storage.
  - SQLite Databases
    - Store structured data in a private database.
  - Network Connection
    - Store data on the web with your own network server.

Expected Screen



ListViewDynamic

HD Wallpaper App

SMemoExports

.books.ebook.pdf.reader

.userReturn

.dianxinfolderstate

.dusd

sample_images

AndroidOptimizer

.PolarisViewer5

iOS videos

apk

Hands on Example -Coding

- activity_main.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/lview1 "
        ></ListView>
</LinearLayout>
```

Hands on Example -Coding

- MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import java.io.File;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView lview=(ListView)findViewById(R.id.lview1);
                // Retrieve the file from the Phone Internal Memory
                //  String path = "/storage/sdcard0/";
                 String path =  "/storage/emulated/0/";
                File f  = new File(path);
        String[] files = f.list();
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(MainActivity.this,android.R.layout.simple_spinner_dropdown_item,files);
        lview.setAdapter(adapter);
    }
}
```

To get the files from your phone internal storage use either one of the following

- **"/storage/sdcard0/"**
- **"/storage/emulated/0/"**

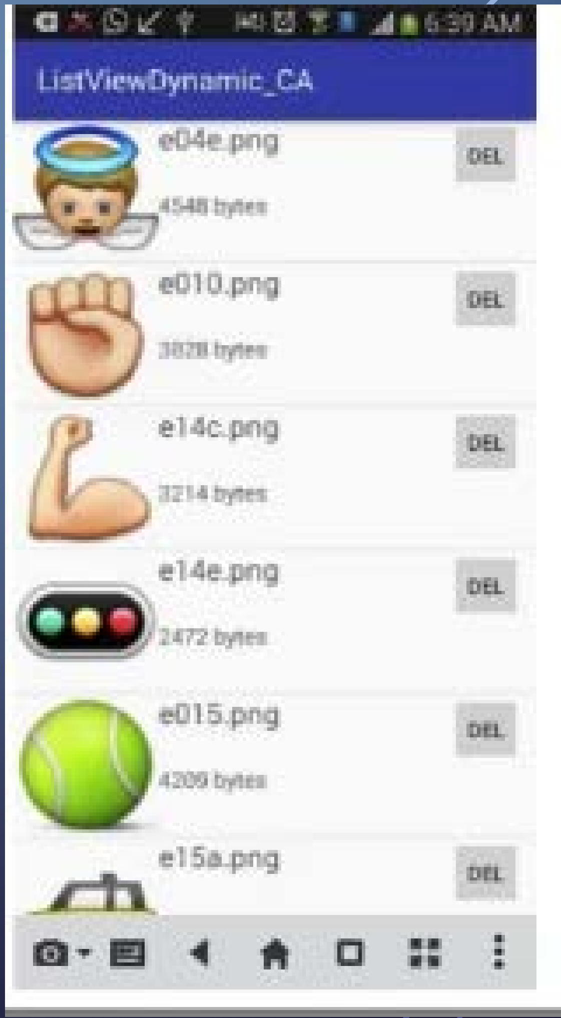
DAY – 2
CUSTOMIZED LISTVIEW

Hands on Example – Customized ListView

- **Problem Requirement :** Display the image with file name, file size and delete button in a each row of ListView. This is called Customized View. User can change the ListView according to their own need. ArrayAdapter we can present only String type of data , if you want to present your own UI on individual , use CustomAdapter.
- Steps to work with CustomAdapter :

1. Create a class with subtype of BaseAdapter. It is an abstract class having 4 abstract methods. Override the following methods in your class.

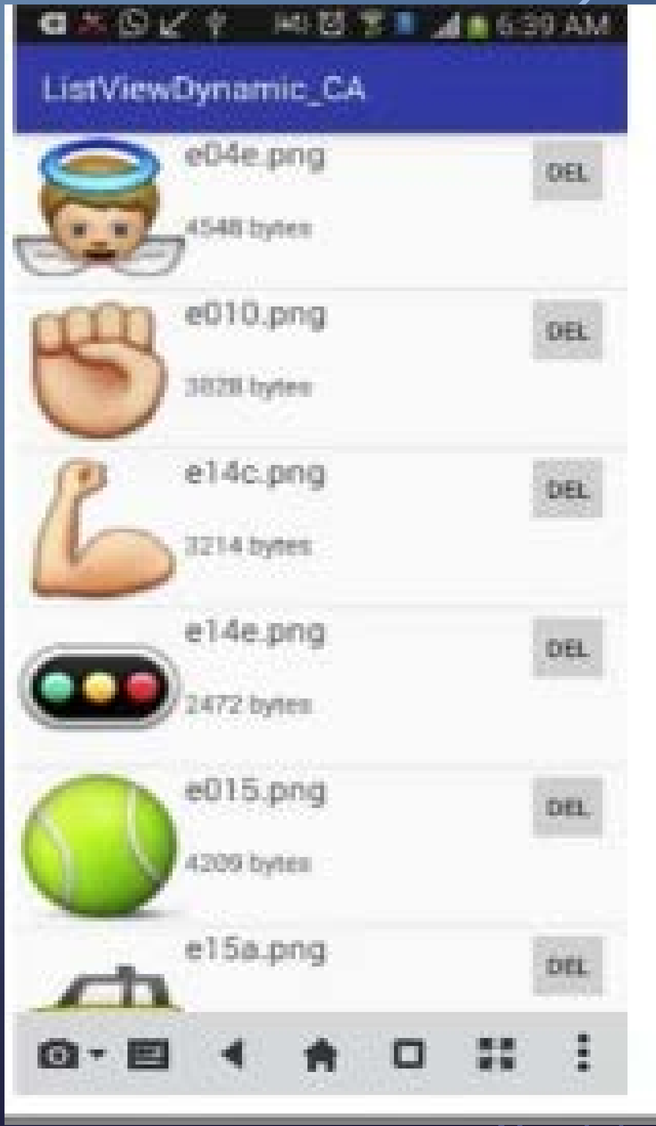
- getCount(), getItem(), getItemId(), getView()
- 2. from Activity class set the custom adapter object to the UI component using
- ui.setAdapter(new CustomAdapter());



Hands on Example – Customized ListView – Coding Step1

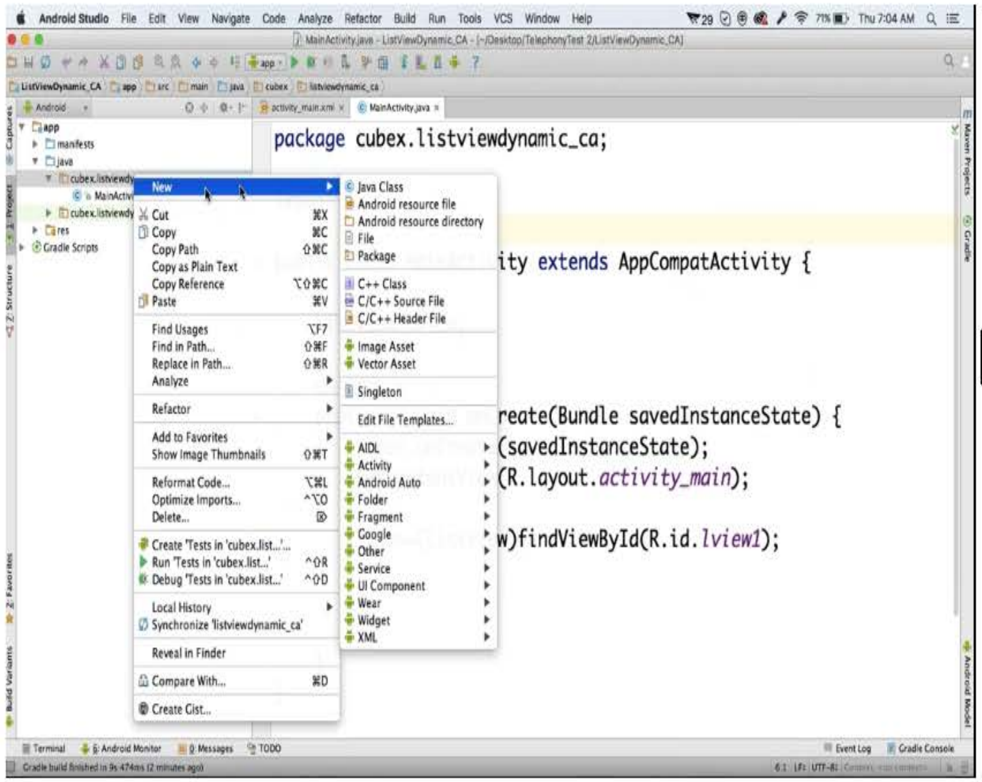
- Step : 1 Design the activity_main.xml with ListView

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/lview1"
        ></ListView>
</LinearLayout>
```



Hands on Example – Customized ListView – Coding Step2

- Step : 2 Create your own Adapter class which extends BaseAdapter and Override the methods. In the previous example we used ArrayAdapter. But now we are customizing the view as per our requirement.
- Create a class MyAdpater by right click on your Activity folder → New→Java Class

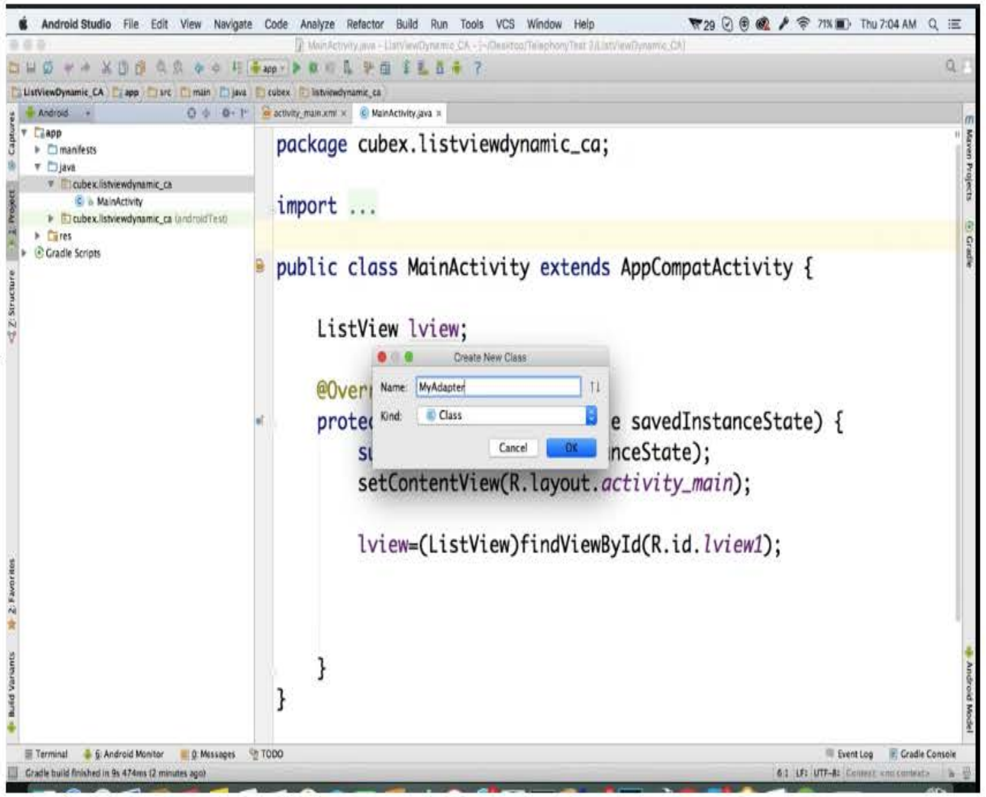


```
package cubex.listviewdynamic_ca;

ity extends AppCompatActivity {

reate(Bundle savedInstanceState) {
(savedInstanceState);
(R.layout.activity_main);

w)findViewById(R.id.lview1);
```



```
package cubex.listviewdynamic_ca;

import ...

public class MainActivity extends AppCompatActivity {

    ListView lview;

    @Over
    protec                          e savedInstanceState) {
        su                          nceState);
        setContentView(R.layout.activity_main);

        lview=(ListView)findViewById(R.id.lview1);

    }
}
```

Hands on Example – Customized ListView – Coding Step2

Structure of the MyAdapter.java

```
public class MyAdapter extends BaseAdapter {
     @Override
      public int getCount() {
        return 0;    }
    @Override
    public Object getItem(int position) {
        return null;
    }
    @Override
    public long getItemId(int position) {
        return 0;
    }
    @Override
     public View getView(int position, View convertView, ViewGroup parent) {
                return null;
    }
}
```

Hands on Example – Customized ListView – Coding Step2

Enhancement of MyAdpater.java step by step code

```
public class MyAdapter extends BaseAdapter {
 /* 1. Specify the path to retrieve the files - Here my specific folder is sample_images
    *     Create a specific folder sample_images on your device internal memory and copy
             some images in that folder
    * 2.  Create a File object by passing the path
    * 3.  Using list() method from the file object to retrieve all images files.
    *     This method returns an array of String */
// We discussed two ways to retrieve from phone internal memory.

    String path="/storage/sdcard0/sample_images/";
    File f=new File(path);
    String[] files=f.list(); // Hold the list of files
    @Override
    // How many values are going to present in the  ListView - which is ListView Count
    public int getCount() {
       return files.length;   // length provide the number of files stored in the array
    }
```

```
// No need to override
   @Override
   public Object getItem(int
position) {
       return null;
   }
// No need to override
   @Override
   public long getItemId(int
position) {
       return 0;
   }
```

Hands on Example – Customized ListView – Coding Step2

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

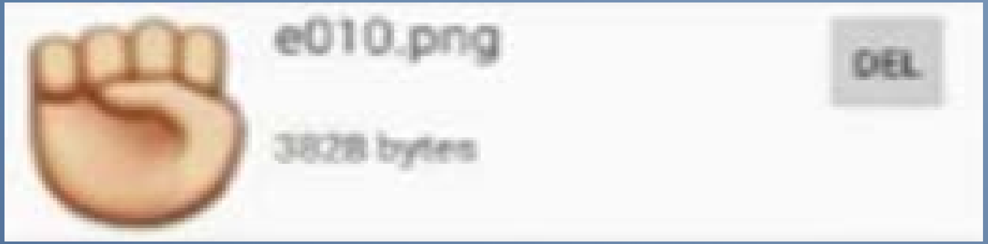
From the above signature of getView(),

- *The return type of this method is View.*
- *View represents in which format you want to present the UI.*
- *To do this create an XML file which customize the user requirements.*
- *Convert the XML file in to the View with the help of LayoutInflater class. It instantiates a layout XML file into its corresponding View objects. It is an Abstract class. So you cannot create an object directly. Get the object using its factory methods.*

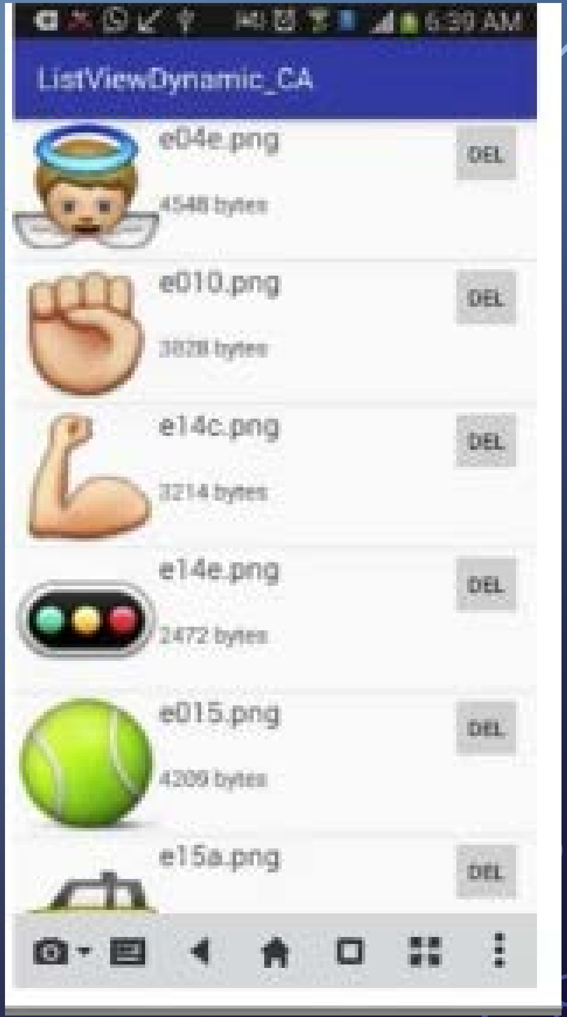
$$\begin{tikzcd}\boxed{\text{XML File}} \arrow[r] & \boxed{\text{LayoutInflater}} \arrow[r] & \boxed{\text{View}}\end{tikzcd}$$

Hands on Example – Customized ListView – Coding Step3

- *Creation of activity_myview.xml*
- *Take a look on the expected output in this image.*



- *Each row in a ListView contains*
  - *One ImageView*
  - *Two TextView, which is used to display the File name and file Size*
  - *One Delete Button.*
- *Design your activity_myview.xml using LinearLayout with the above specified components.*



Hands on Example – Customized ListView – Coding Step3

- *Creation of activity_myview.xml : Click res → layout(Right Click) → New → Layout Resource File*

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <ImageView
      android:layout_width="0dp"
      android:layout_height="100dp"
      android:layout_weight="0.2"
      android:src="@drawable/twitter_logo"
      android:id="@+id/lview"
      />
     <LinearLayout
        android:layout_width="0dp"
        android:layout_height="100dp"
        android:layout_weight="0.6"
        android:orientation="vertical"
        >
<TextView
 android:layout_width="match_parent"
  android:layout_height="0dp"
  android:layout_weight="0.5"
  android:text="File Name"
  android:textSize="20sp"
  android:textColor="#FF0000"
  android:textStyle="bold"
  android:gravity="center"
  android:id="@+id/tv1"
  />
<TextView
 android:layout_width="match_parent"
  android:layout_height="0dp"
  android:layout_weight="0.5"
  android:text="File Size"
  android:textSize="20sp"
  android:textColor="#0000FF"
  android:textStyle="bold"
  android:gravity="center"
  android:id="@+id/tv2"
 />
</LinearLayout>
<Button
  android:layout_width="0dp"
  android:layout_height="wrap_content"
  android:layout_weight="0.2"
  android:text="Del"
  android:id="@+id/bt1“
/>
</LinearLayout>
```

Hands on Example – Customized ListView – Coding Step4

- MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    // To access without object reference in the MyAdaper.java
     static MainActivity mainActivity;
    ListView lview;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mainActivity = this; // Initialize the current activity Reference
        lview=(ListView)findViewById(R.id.lview1);
        lview.setAdapter(new MyAdapter()); // Specify the object of your Custom Adapter
    }
// To refresh the ListView once elements are removed by clicking the Del button
  public  void refresh(){
     lview.setAdapter(new MyAdapter());
  }
}
```

Hands on Example – Customized ListView – Coding Step2

- Implementation of getView() method in MyAdapter.java.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    // Obtains the LayoutInflater from the given context, in which activity class you want to present
    LayoutInflater inflater = LayoutInflater.from(MainActivity.mainActivity);

// Convert the XML file into View object
View view = inflater.inflate(R.layout.activity_myview,null);

}
```

Inflate() method

View inflate (int resource, ViewGroup root)

Inflate a new view hierarchy from the specified xml resource.

Throws InflateException if there is an error.

Parameters	
resource	int: ID for an XML layout resource to load (e.g., R.layout.main_page)
root	ViewGroup: Optional view to be the parent of the generated hierarchy.

Returns	
View	The root View of the inflated hierarchy. If root was supplied, this is the root View; otherwise it is the root of the inflated XML file.

Hands on Example – Customized ListView – Coding Step2

- Implementation of getView() method in MyAdapter.java.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
 // Obtains the LayoutInflater from the given context, in which activity class you want to present
     LayoutInflater inflater = LayoutInflater.from(MainActivity.mainActivity);

   // Convert the XML file into View object
     View view = inflater.inflate(R.layout.activity_myview,null);

  // Configure the Id for individual UI from the activity_myview.xml
     ImageView imageView = (ImageView)view.findViewById(R.id.lview);
     TextView fname = (TextView)view.findViewById(R.id.tv1);
     TextView fsize = (TextView)view.findViewById(R.id.tv2);
     Button delete = (Button)view.findViewById(R.id.bt1);

  /* To display the actual image in each row,
  already we have the actual path and retrieve each image using position
  and then make the String into File.
  Set the image to the ImageView UI using Uri.fromFile() method by passing the File object*/
          String new_path = path + files[position];
          final File new_file = new File(new_path);
         imageView.setImageURI(Uri.fromFile(new_file));
```

Hands on Example – Customized ListView – Coding Step2

- Implementation of getView() method in MyAdapter.java.

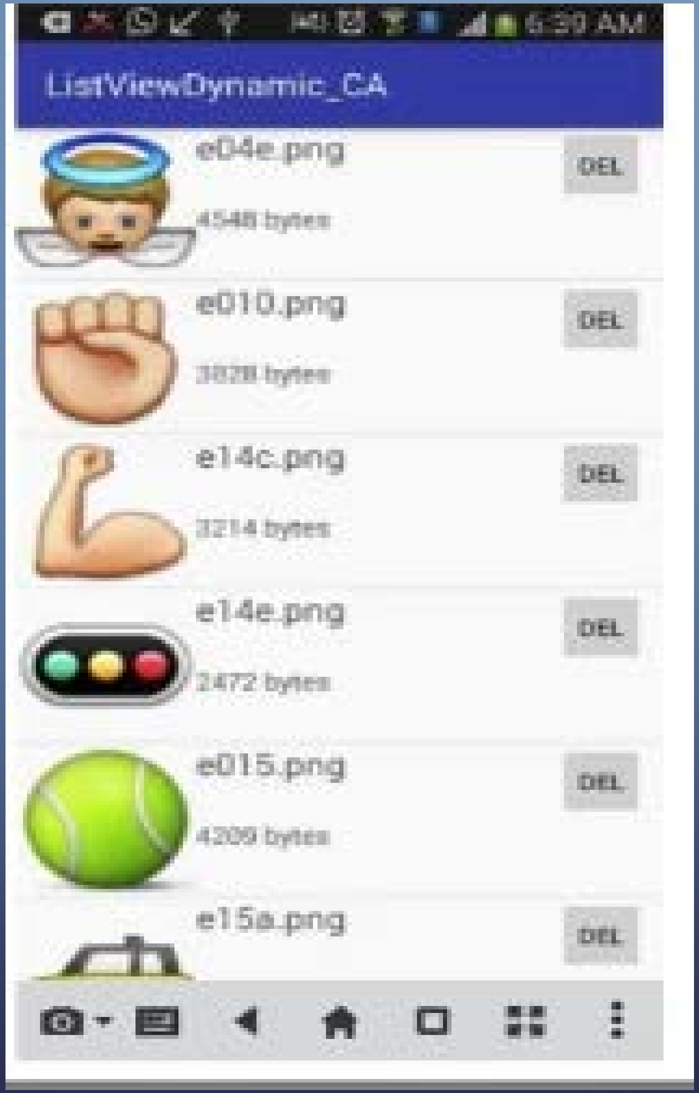
```
// Set the file name and file size to the TextView UI
    fname.setText(files[position]); // Retrieve the file name
    fsize.setText(new_file.length() + " bytes"); // Retrieve the file size

// Remove the file once click the Del button by implementing OnClickListener()

      delete.setOnClickListener(new View.OnClickListener() {
      @Override
       public void onClick(View v) {
                    new_file.delete();  //  helps to remove the file
             MainActivity.mainActivity.refresh(); // Refresh the ListView once the file is removed
      }
   });
   return view; // Return the View
  }
}
```

Hands on Example – Customized ListView – Step 5

Run your App to get the Expected result.



Other Views

- We learned ListView in this course
- Like that Android supports several views which we are not covered
  - GridView - This layout displays a scrolling grid consisting of rows and columns
  - GalleryView - Provides layout to view all the images in a gallery view
  - RecyclerView - **RecyclerView** is like traditional ListView widget, but with more flexibility to customizes and optimized to work with larger datasets.
  - SurfaceView- Provides a dedicated drawing surface embedded inside of a view hierarchy. To handle Graphics and Animation.