Android ListView

Android ListView is a view group that displays a list of scrollable items. If you have a date which is repeated in the form of a collection or list, the listview is the best User Interface element to use. ListView helps you in displaying repeating data in the form of a scrollable list. Users can then select any listitem by clicking on it.

ListView is widely used in Android Applications. A simple example of ListView is your contact book, where you have a list of your contacts displayed in a ListView.

Let's take an example:

Have a look at the screenshot of the twitter app described above.

All the rows are similar, and the only thing which changes is the tweet content. Such situation drives us to use Android ListView.

Using Android ListView

Android provides ListView or ExpandableListView. ExpandableListView contains list items which can be expanded. To use a listview in Android you can drag and drop the listview control from the palette to your UI.

Here is how your Android listview code looks:

```
<ListView
    android:id="@+id/listView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
</ListView>
```

ListItem

An Android listview is made from a group of list items. List items are individual rows in listview where the data will be displayed. Any data in listview is displayed only through listItem. Consider Listview as scrollable group of list items.

List items are just layouts in a separate layout file. Let us understand the following example.

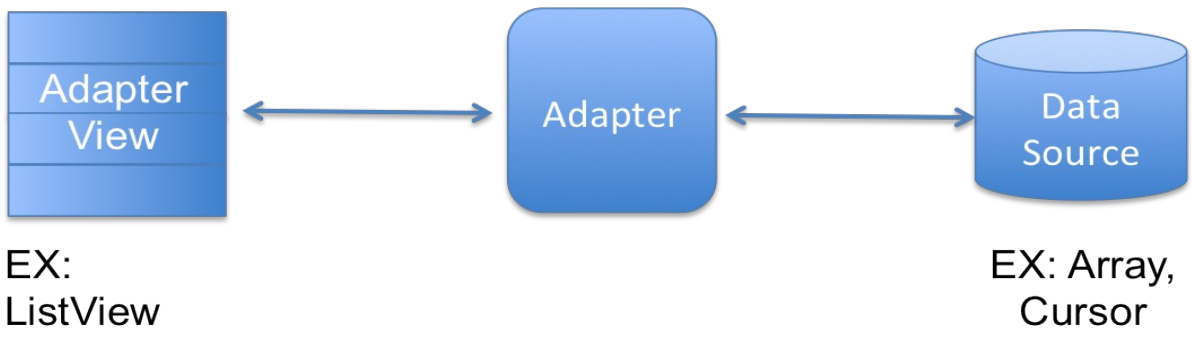
Here we can see a listitem for the twitter application. This list Item is arranged in a Relative layout with images and multiple text views aligned to each other. This is how an Android listview is designed.

Once we have the listitem, we bind the listview to the Adapter and then use list items to display the data in listview.

Adapter

Android Adapter is a bridge between the View (e.g. ListView) and the underlying data for that view. An adapter manages the data and adapts the data to the individual rows (listItems) of the view.

We bind the adapter with Android listview via *setAdapter* method. Now, Let us see how adapter works with the help of the following image.



As stated earlier, Adapters act as a bridge to the views. To interact with the view, adapters call the getView() method which returns a view for each item within the adapter view. This is a listitem which we have seen earlier. The layout format and the corresponding data for an item within the adapter view are set in the getView() method.

Once we have a reference to the view, we can get the data from the data source either in an Array list or a cursor. We can then bind the data to the view items. All this is done in getView Method. In coming sections, we will look as to how we can achieve this in our code.

Typically you do not have to deal directly with the Adapter class. Android provides you commonly used Adapters like :

Array Adapter Cursor Adapter *Simple Cursor Adapter

Most of the time we tend to write our own Adapter by extending *BaseAdapter*

Let's see Adapters in Action now.

Basic ListView with ArrayAdapter

ArrayAdapter

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

By default, ArrayAdapter expects a Layout with single TextView. If you want to use more complex views, please avoid ArrayAdapter and use custom adapter

Android ListView Example (ArrayAdapter)

For ArrayAdapter to work you need to have an array of data, let's create an Array of Code Learn Tutorial Chapter:

```
    String[] codeLearnChapters = new String[] { "Android
Introduction","Android Setup/Installation","Android Hello World","Android
Layouts/Viewgroups","Android Activity & Lifecycle","Intents in Android"};
```

Since we have the data now, let's create a new Object of ArrayAdapter.

```
    ArrayAdapter<String> codeLearnArrayAdapter = new
ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
codeLearnChapters);
```

Observe the parameters for creating ArrayAdapter:

1. First Parameter is context

2. Second Parameter is the Layout, which this ArrayAdapter will use to bind the data from codeLearnChapters Array. Android comes pre bundled with some common layout which you can refer with android.R.layout. Here we are using - simple_list_item_1, which is just a simple text view.

3. Third parameter is the data, in this case the String Array.

Now that we have ArrayAdapter created, let's bind the Adapter with the listview.

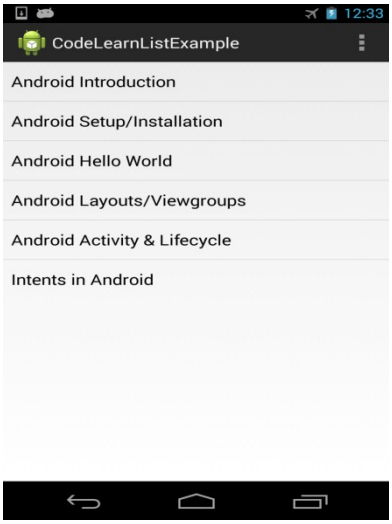
Let's get the reference of listview

```
ListView codeLearnLessons = (ListView)findViewById(R.id.listView1);
```

Once you have a reference of listview, you can just call *setAdapter* method by passing the ArrayAdapter to be bound.

```
codeLearnLessons.setAdapter(codeLearnArrayAdapter);
```

With these simple 4 lines of code, you created your first Android listview. If you run the application ,you will see the array adapter in action. This is how it will look:



Custom ListView with BaseAdapter

We have seen above what we can do with ArrayAdapter. But for most use cases you will have a complex layout for list items as below

This kind of complex list items can be bound with BaseAdapter. With base Adapter you can practically build any layout which you want in your Android listview.

BaseAdapter

BaseAdapter is a common base class of a general implementation for an Adapter that can be used in ListView. BaseAdapter can be extended to create a custom Adapter to suit your need of building a custom listitem.

ArrayAdapter is also an implementation of BaseAdapter

Android ListView Example (Custom Adapter)

Now since we understand the concepts of BaseAdapter, Let's Create our own Custom Adapter.

To create a custom adapter all you need to do is create a new class then extends BaseAdapter, for example

```
public class CodeLearnAdapter extends BaseAdapter
```

With this simple line of code, you can create your own adapter. Once you extend the BaseAdapter, it will automatically ask you to implement the required method. Just use Eclipse, add required methods to be implemented and it will automatically create the code for you.

This is how your Custom Adapter will look like

```
public class CodeLearnAdapter extends BaseAdapter {

    @Override
    public int getCount() {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public Object getItem(int arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public long getItemId(int arg0) {
        // TODO Auto-generated method stub
        return arg0;
    }

    @Override
    public View getView(int arg0, View arg1, ViewGroup arg2) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Understanding Custom Adapter Methods

Let's go step by step to understand each method of the custom adapter.

- getCount() : This method tells the listview the number of rows it will require. This count can come from your data source. It can be the size of your Data Source. If you have your datasource as a list of objects, this value will be the size of the list.

- Object getItem(int arg0): We have 2 method implementation of getItem. This method returns an object. This method helps ListView to get data for each row. The parameter passed is the row number starting from 0. In our List of Objects, this method will return the object at the passed index.

- long getItemId(int arg0) : You can ignore this method. It just returns the same value as passed. This in general helps ListView to map its rows to the data set elements.

- getView : This is the most important method. This method will be called to get the View for each row. This is the method where we can use our custom listitem and bind it with the data. The fist argument passed to getView is the listview item position ie row number. The second parameter is recycled view reference(as we know listview recycles a view, you can confirm through this parameter). Third parameter is the parent to which this view will get attached to.

Now we understand all the methods of custom adapter. Let's create our own Custom ListView

Custom ListItem

First Step to have a custom ListView is to have a custom listitem. Let's create a custom listitem like this

This is similar to the one used in the twitter app.

This is the code for listitem.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
     >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"

        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="23dp"
        android:layout_marginTop="10dp"
        android:layout_toRightOf="@+id/imageView1"
        android:text="CodeLearn Chapter 1"
```

```
            android:textSize="16sp" />

        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignBottom="@+id/imageView1"
            android:layout_alignLeft="@+id/textView1"
            android:text="Description" />

    </RelativeLayout>
```

Data Source

Now since we have the custom listitem, we only need data. For this example project, we will just use a List of codeLearnChapter Objects which will have a chapter name and chapter description. Here is how the data structure of codeLearnChapter object looks like

```
    public class codeLearnChapter {
            String chapterName;
            String chapterDescription;
        }
```

In the sample code, we have created a simple method getDataForListView which returns a populated list of codeLearnChapter objects. This is how the method looks

```
    public List<codeLearnChapter> getDataForListView()
    {
        List<codeLearnChapter> codeLearnChaptersList = new
ArrayList<codeLearnChapter>();

        for(int i=0;i<10;i++)
        {

            codeLearnChapter chapter = new codeLearnChapter();
            chapter.chapterName = "Chapter "+i;
            chapter.chapterDescription = "This is description for chapter "+i;
            codeLearnChaptersList.add(chapter);
        }

        return codeLearnChaptersList;

    }
```

Populating Custom Adapter

We now have a listitem, and also the data. Let's Create our own adapter. We have already created a custom adapter in the previous section, let's make some changes and put code in the methods.

Getting the Data Let's get the data into our custom Adapter by calling getDataForListView method. We will create a new variable inside our custom adapter.

```
    List<codeLearnChapter> codeLearnChapterList = getDataForListView();
```

getCount Now we have the data, let's add code in getCount method. As discussed earlier, this method tells ListView how many rows it will have. In our case, this is the size of codeLearnChapterList. So let's return this size.

```
    return codeLearnChapterList.size();
```

Object getItem This method returns the object of the datasource at specified position. In our case, the object is of type codeLearnChapter. Let's modify the return type, and add a return statement.

```
    @Override
        public codeLearnChapter getItem(int arg0) {
            // TODO Auto-generated method stub
            return codeLearnChapterList.get(arg0);
        }
```

long getItemId Leave this method as it is, just make sure it is returning the same value that is passed.

```
    return arg0;
```

getView

Now this is our main method where we will bind the listitem to the ListView. Since we want to use our own listitem, let's get the reference of our custom listitem. To add any new layout to an existing layout, you need to inflate the other layout. To inflate a layout you need to get the reference of LayoutInflater.

```
    LayoutInflater inflater = (LayoutInflater)
ListViewWithBaseAdapter.this.getSystemService(Context.LAYOUT_INFLATER_SERVICE)
;
```

In this code, you just called getSystemService method by passing Context.LAYOUT_INFLATER_SERVICE. Getsystem service method is present in context. To get the reference of context, you can call **this** pointer on Activity,

Once you have the layout inflater object, you just need to call inflate method on it.

```
     View rowView = inflater.inflate(R.layout.listitem, arg2,false);
```

This inflates method takes 3 parameters, the layout which you want to add, the parent where the layout is being added , and third a boolean to attach the root to it's parent.

Now we have a reference of listitem layout, but we know that listview recycles the views. Lets check the second parameter of getView to see if we already have te reference of listitem.

```
    if(arg1==null)
    {
        LayoutInflater inflater = (LayoutInflater)
ListViewWithBaseAdapter.this.getSystemService(Context.LAYOUT_INFLATER_SERVICE)
;
        arg1 = inflater.inflate(R.layout.listitem, arg2,false);
    }
```

With this simple check, we can save the overhead of inflating a new view for each row. This will make our listview faster.

Now we have the listitem, let's bind the data with the listview item.

First let's get the reference of listitem elements, in this case we want the reference of both the text views

```
    TextView chapterName = (TextView)arg1.findViewById(R.id.textView1);
    TextView chapterDesc = (TextView)arg1.findViewById(R.id.textView2);
```

Now Let's get the Data We have the list of codeLearnChapterList, let's call a get method on the list by passing the row id which is coming as 1st parameter of getView.

```
    codeLearnChapter chapter = codeLearnChapterList.get(arg0);
```

Binding Data Now let's get data from chapter object and bind it to the textview.

```
    chapterName.setText(chapter.chapterName);
    chapterDesc.setText(chapter.chapterDescription);
```

Now that we have done everything, time to just return the view

```
    return arg1;
```

With this we have our custom Adapter ready to get binded to the listview.

This is how the complete custom adapter looks :

```
    public class CodeLearnAdapter extends BaseAdapter {

        List<codeLearnChapter> codeLearnChapterList = getDataForListView();
        @Override
        public int getCount() {
            // TODO Auto-generated method stub
            return codeLearnChapterList.size();
        }

        @Override
        public codeLearnChapter getItem(int arg0) {
            // TODO Auto-generated method stub
            return codeLearnChapterList.get(arg0);
        }

        @Override
        public long getItemId(int arg0) {
            // TODO Auto-generated method stub
            return arg0;
        }

        @Override
        public View getView(int arg0, View arg1, ViewGroup arg2) {

            if(arg1==null)
```

```
            {
                LayoutInflater inflater = (LayoutInflater)
ListViewWithBaseAdapter.this.getSystemService(Context.LAYOUT_INFLATER_SERVICE)
;
                arg1 = inflater.inflate(R.layout.listitem, arg2,false);
            }

            TextView chapterName =
(TextView)arg1.findViewById(R.id.textView1);
            TextView chapterDesc =
(TextView)arg1.findViewById(R.id.textView2);

            codeLearnChapter chapter = codeLearnChapterList.get(arg0);

            chapterName.setText(chapter.chapterName);
            chapterDesc.setText(chapter.chapterDescription);

            return arg1;
        }

    }
```

Binding Adapter with ListView

First Let us create the object of our custom adapter.

```
    CodeLearnAdapter chapterListAdapter = new CodeLearnAdapter();
```

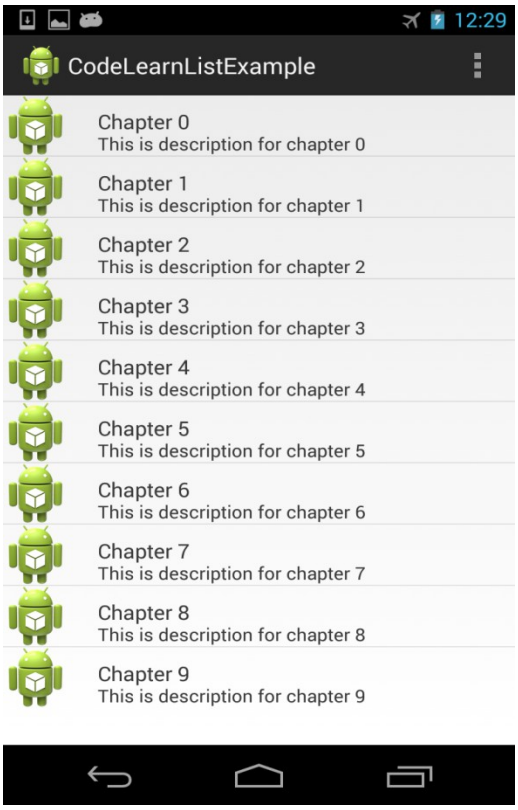
Now let us take the reference of the listview

```
    ListView codeLearnLessons = (ListView)findViewById(R.id.listView1);
```

Now Let us bind the Adapter with ListView by calling setAdapter Method.

```
    codeLearnLessons.setAdapter(chapterListAdapter);
```

This is how the ListView will look when you run it:



ListActivity

In both of our examples, we have extended the Activity class and bound the listview. Though this is a simple way to do most of your work in android, android also comes with specialized Activities which can help you reduce the overall length of the code you write. These activities give you shortcut methods to make your development easier and faster.

ListActivity is one such example of specialized activity which android offers. ListActivity is a subclass of Activity which makes creating, binding and using ListView simple. Before going deep into ListActivity, here are some of the advantages of using ListActivity.

- List Activity Binds with a listview by default, no need to explicitly get the ListView object
- ListActvity holds a ListView object which can be bound by calling setAdapter anywhere in the code.
- Since it holds the reference of the listview, you can directly add the click listener.

Let's See how we can use ListActivity in our CustomAdapter Example. To do this, we will create a new Activity which will extend ListActivity.

XML

The first thing that will be changed while we use the ListActivity is the listview id in layout. ListActivity always expects the layout which you bind it with to have a listview with id *@android:id/list*.

Why is this required ? If you recall, earlier we stated that ListActivity automatically holds the reference of a ListView. This is achieved via a common naming convention of the ListView whose object is instantiated in list activity automatically.

Go to list_view_list_activity.xml in our example project to see ListView with a new id.

Extends

To use ListActiviy just extend it in your class instead of an Activity. This will make the most of the boilerplate code available for you to use.

Binding Data

In Case of ListActivity, you have the advantage of binding the adapter directly by calling *setListAdapter* method.

Here is how you bind data in usual case, where you get the listview object and then call setAdapter on it.

```
    ListView codeLearnLessons = (ListView)findViewById(R.id.listView1);
    codeLearnLessons.setAdapter(chapterListAdapter);
```

In case of ListActivity, you just need to call setListAdapter by passing the adapter:

```
    setListAdapter(chapterListAdapter);
```

Click Listener

ListActivity provides you with a default method onListItemClick. This will be invoked every time someone clicks on a listview row. With this method, you don't need to set an explicit listener on your ListView.