Title: Adapter Views, List Views & Custom Views

1. Adapter Views: AdapterView is a subclass of ViewGroup class. AdapterViews are named so because an Adapter class object decides the content and view inside them.

**Adapters:** Adapters act as data provider for Adapter Views. They get data from the underlying data source. Adapters are also responsible for setting up the Views for different items.

Adapters are like an outer shell on our list, grid view that hold actual information about our list. ListView and GridView are just views and they cannot manipulate the data inside them, for this task, we need adapters who basically adapt a list to our needs. It controls what data goes into the list and how each item of a list looks. It contains the actual logic code.

Some of the example adapter views are ListView, GridView, Recycler View and Gallery etc. We will look at ListView in further details.

1. List Views: Listviews are used to show data items in a vertically scrolling list. Here are the steps to make a ListView work. You would have seen list views within a range of different applications. They are one of the most used components.
   1. Adding ListView to UI: You can simply drag and drop a listview into the preview subscreen of Graphical Layout tab. XML code for the same looks like this.

<ListView

android:id=*"@+id/listView1"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"*

android:layout\_alignParentLeft=*"true"*

android:layout\_alignParentTop=*"true"* >

</ListView>

This will add a list view completing filling its parent’s view.

* 1. Array Adapter: ArrayAdapter is one of the simplest Adapters. You can simplify provide it with a List of items and ArrayAdapter will show these in the ListView. Here are the steps to get a simple ArrayAdapter working.
     1. Construction: ArrayAdapter has a bunch of constructors that can be used to initialize an object. Lets look at one example.

ArrayAdapter<String> arrayAdapter = **new** ArrayAdapter<String> (**this**, android.R.layout.*simple\_list\_item\_1*,datalist);

Here we have initialized an ArrayAdapter, which uses a simple list item layout (predefined in android library) to create views for each item. Data is taken from datalist object, which is an object of List type.

* + 1. Connecting Adapter and List View: Now we just need to call the setAdapter function on the ListView and pass the just created adapter as the argument. Here is the example code:

ListView listview = (ListView)findViewById(R.id.*listView1*);

listview.setAdapter(arrayAdapter);

After this step you should be able to see the data in your list view.

* 1. Handling item clicks: Great, seems like we can show list of data to user now but most of the times we don’t just want to show the data to user, we want to allow the user to interact with these items to either edit them or see more details. In order to achieve this we need to use setOnItemClickListener for the listview. The listener object must have implemented the OnItemClickListener interface.

Here is some example code:

listview.setOnItemClickListener(**new** OnItemClickListener() {

@Override

**public** **void** onItemClick(AdapterView<?> adapter, View view, **int** position,**long** lid) {

Toast.*makeText*(getActivity().

getApplicationContext(), "Item Clicked",

Toast.*LENGTH\_LONG*).show();

}

});

In this code we are simply showing a Toast message to the user whenever an item is pressed.

* 1. Accessing listview items: Most of the times we would like to do something more than showing a generic toast to the user. We would need to access the item being pressed and use that information. You can use the position passed by system in the onItemClick method and pass that to getItemAtPosition function of the adapter to get the item clicked. Example code:

String item = (String) adapter.getItemAtPosition(position)

As we can see the adapter contained the actual information about the item at a particular position.

* 1. Changing listview Content: You can update ListView’s content by simply changing the underlying datalist object and calling notifyDataSetChanged method of the adapter.

1. Custom Views: Most of the times the default implementation won’t be enough and you would like to show more complicated layouts inside standard UI elements. Before doing so we need to learn about layout inflators.
   1. LayoutInflator: Layout inflators are used to read layout XML files and create the corresponding views. You should not create a LayoutInflator yourself, instead you should use one provided by your activity. Example code to get a LayoutInflator and creating a view.

LayoutInflater inflator = getLayoutInflater();

View output = inflator.inflate(R.layout.*a\_layout*, **null**);

Second argument to inflate function is a ViewGroup to which the created view should be attached.

* 1. Toasts: When we use makeText function of the Toast class, we are getting a toast object with a default view. Instead of this you can use a constructor to get a Toast object and assign it a view by using setView method. You can get the View object by using a LayoutInflator.

Toast toast = **new** Toast(getApplicationContext());

toast.setDuration(Toast.*LENGTH\_LONG*);

toast.setView(output);

toast.show();

* 1. Alert Dialogs: AlertDialogs are used to get a confirmation from the user or get a quick input from the user. AlertDialog class has a nested class AlertDialog.Builder, which can be used to create alert dialogs. Here are example code for creating an AlertDialog
     1. Getting builder object

AlertDialog.Builder b = **new** AlertDialog.Builder (this);

* + 1. Title: Setting title of the dialog

b.setTitle("Add New Ticker");

* + 1. View: Get the View object by using the inflator and use following code to setup the custom view.

b.setView(output);

* + 1. Buttons: Alert Dialogs support upto three buttons. These are positive button, neutral button and negative button. You can use setButton function on the AlertDialog to set any of these buttons. You can also use the setPositiveButton, setNeutralButton and setNegativeButton functions on the builder class. Example code for setting up positive button.

b.setPositiveButton("Add", **new**

OnClickListener() {

@Override

**public** **void** onClick(DialogInterface arg0,

**int** arg1) {

// Code to handle positive button click

}

* + 1. Creating and Showing the AlertDialog from the Builder:

b.create().show();

1. Custom List View: Many a times we would need to have a list view with items showing more than just a textview. In order to achieve this we will have to subclass the ArrayAdapter class and use an object of the custom class as adapter with our ListView. E.g. in the following example I am going to assume that we need to show company names and their stock prices in the ListView
   1. Custom Adapter: Create a new class and use ArrayAdapter as the base class. For this class we know that the list is going to have objects of class Company, so we can set the base class’s generic argument as Company.

**public** **class** CustomArrayAdapter **extends** ArrayAdapter<Company>

Here are some of the functions that you should know about:

* + 1. Constructor: You would have to implement one constructor for the custom adapter. Implement one, which looks closest to your needs.
    2. getView: In order to provide your own layout for list items you will have to implement this method. Android will call this function every time an item enters the screen. Here is the prototype for the function:

**public** View getView (**int** position, View convertView, ViewGroup parent)

Here is what different arguments mean:

* + - 1. Position: This is the position of the item for which you need to return the view.
      2. Convert View: Most of the times your list items will be generated using the same layout file. We can achieve far better performance by recycling the views instead of creating a new view each time this function gets called. In order to enable this Android sends you a view of an item, which has been scrolled out of the screen. You can just reset the data inside the old view and use it for the current item.
      3. Parent: You wouldn’t be using this argument most of the times. This is the parent view inside which the requested view is going to get added to.

As we can see the return type of this function is a View, therefore we need to return a view from it, which is this view? This function returns the view we want each of our list’s item to have.

* + 1. getViewTypeCount: This function should be used when you have more than one type of layouts for list items. By providing this information you are telling the system that we need to have different sets of views for recycling. i.e. you cant use view type 1 for a item which requires view type 2.
    2. getItemViewType : This function provides the viewtype to be used for a given position.

1. Recycler View:

The RecyclerView is a new ViewGroup that is prepared to render any adapter-based view in a similar way. It is supossed to be the successor of ListView and GridView, and it can be found in the latest support-v7 version.

If you want to use a RecyclerView, you will need to feel comfortable with three elements:

– RecyclerView.Adapter

– LayoutManager

– ItemAnimator

First of all, before getting started, let us include Recycler View in our project:

compile 'com.android.support:recyclerview-v7:+'

Now add the recycler view to xml file:

<android.support.v7.widget.RecyclerView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:id="@+id/recycler\_view"

/>

Now let us look at every element one by one:

1. Adapter:

It adds two new methods like onCreateViewHolder() and onBindViewHolder() to organize the code. You must override these two methods for inflate the view and to bind data to the view.

public class MyRecyclerAdapter extends RecyclerView.Adapter<MyRecyclerAdapter.ViewHolder> {

private List<ViewModel> items;

private int itemLayout;

public MyRecyclerAdapter(List<ViewModel> items, int itemLayout) {

this.items = items;

this.itemLayout = itemLayout;

}

@Override public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

View v = LayoutInflater.from(parent.getContext()).inflate(itemLayout, parent, false);

return new ViewHolder(v);

}

@Override public void onBindViewHolder(ViewHolder holder, int position) {

ViewModel item = items.get(position);

holder.text.setText(item.getText());

holder.image.setImageBitmap(null);

Picasso.with(holder.image.getContext()).cancelRequest(holder.image);

Picasso.with(holder.image.getContext()).load(item.getImage()).into(holder.image);

holder.itemView.setTag(item);

}

@Override public int getItemCount() {

return items.size();

}

public static class ViewHolder extends RecyclerView.ViewHolder {

public ImageView image;

public TextView text;

public ViewHolder(View itemView) {

super(itemView);

image = (ImageView) itemView.findViewById(R.id.image);

text = (TextView) itemView.findViewById(R.id.text);

}

}

}

//This is for sample, I will add later a code that will be used in the class.

b) Layout Manager:

This option decides how our recycler view is to look, whose characteristic to adapt. The options available are:

1. [LinearLayoutManager](http://developer.android.com/reference/android/support/v7/widget/LinearLayoutManager.html) shows items in a vertical or horizontal scrolling list.
2. [GridLayoutManager](http://developer.android.com/reference/android/support/v7/widget/GridLayoutManager.html) shows items in a grid.
3. [StaggeredGridLayoutManager](http://developer.android.com/reference/android/support/v7/widget/StaggeredGridLayoutManager.html) shows items in a staggered grid.

Example code:

recyclerView.setLayoutManager(new LinearLayoutManager(this));

### c) ItemAnimator

ItemAnimator will animate ViewGroup modifications that are notified to adapter. Basically it will automatically animate adding and removing items. That’s not an easy class either, and we find a *DefaultItemAnimator* that works quite well.

Code Example:

recyclerView.setItemAnimator(new DefaultItemAnimator());

# d) Handle RecyclerView Click Event

A click event in recycler view is not as easy as in ListView, slightly tricky but there is a workaround that can be implemented in the adapter itself:

Put the following code in onBindViewHolder():

customViewHolder.textView.setOnClickListener(clickListener);  
 customViewHolder.imageView.setOnClickListener(clickListener);  
  
 customViewHolder.textView.setTag(customViewHolder);  
 customViewHolder.imageView.setTag(customViewHolder);

Add declare the clickListener variable as follows:

View.OnClickListener clickListener = new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 CustomViewHolder holder = (CustomViewHolder) view.getTag();  
 int position = holder.getPosition();  
  
 FeedItem feedItem = feedItemList.get(position);  
 Toast.makeText(mContext, feedItem.getTitle(), Toast.LENGTH\_SHORT).show();  
 }  
};

1. Homework
   * 1. Build an Android application for showing to do items to user. Add following features.
        1. Show the list of todo’s to the user in a list view. Show the title and deadline to user.
        2. Add a “Create new” button and onClick of this function should open a custom alert dialog, which allows users to enter title, description and deadline. Add a positive button and a negative button.
        3. Update the List View with new item when user completes adding a new todo.
     2. Read about Grid views and implement a layout using Grid view.