ListView and RecyclerView

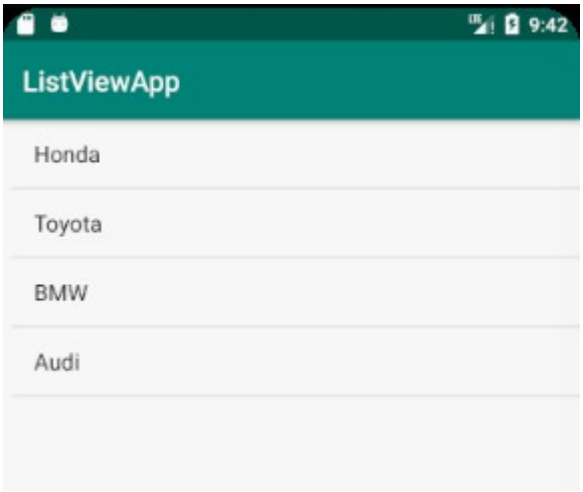
ListView and RecyclerView

In this Lecture, you will learn:

- Displaying lists with ListView
- The Adapter class
- Displaying lists with RecyclerView
- Difference between ListView and RecyclerView

ListView and RecyclerView

ListView:
The ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.



An adapter is a bridge between UI components and the data source that fill data into UI Component.

The ListView and GridView are subclasses are populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

Android provides several subclasses of Adapter to retrieve different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adapters are ArrayAdapter, Base Adapter, CursorAdapter.

ListView Attributes

android:id	This is the ID which uniquely identifies the layout.
android:divider	This is drawable or color to draw between list items.
android:dividerHeight	This specifies height of the divider.
android:entries	Specifies reference to an array that will populate the ListView.
android:footerDividersEnabled	When set to false, the ListView will not draw the divider before each footer view. The default value is true.
android:headerDividersEnabled	When set to false, the ListView will not draw the divider after each header view. The default value is true.

ListView and RecyclerView

ArrayAdapter
You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a TextView. For an array of strings to display in a ListView, initialize a new ArrayAdapter using a constructor to specify the layout for each string and the string array.

ArrayAdapter adapter =
new ArrayAdapter<String>(this, R.layout.ListViewRow, StringArray);

First argument, is the context.
Second argument is layout defined in XML file, having TextView for each string in the array.
Last argument is an array of strings to be populated in the text view.

With array adapter created, call setAdapter() on your ListView object:
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);

Define a ListView under res/layout directory in an XML file (activity_main.xml file for example).

Manufacturers Lists Example App:

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    ListView listView;
    TextView textView;
    String[] listItems;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listView = findViewById(R.id.listView1);
        textView = findViewById(R.id.textViewManufacturer);
        listItems = new String[] {"Honda", "Toyota", "BMW"};

        final ArrayAdapter<String> adapter
                = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1,
                android.R.id.text1,
                listItems);
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(
                new AdapterView.OnItemClickListener() {
                    @Override
```

```
                    public void onItemClick(AdapterView<?> parent,
                                View view, int position, long id) {
                        Toast.makeText(MainActivity.this,
                                adapter.getItem(position),
                                Toast.LENGTH_LONG).show();
                    }
                });
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout  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">
    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="8dp"
        android:layout_marginTop="0dp" />
</RelativeLayout>
```

list_row.xml

```
<?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">
    <TextView
        android:id="@+id/textViewManufacturer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

SimpleCursorAdapter

You can use this adapter when your data source is a database Cursor. Specify a layout to use for each row in the Cursor and columns in the Cursor to be inserted into which views of the layout.

If you want to create a list of people's names and phone numbers, you can perform a query that returns a Cursor containing a row for each person and columns for the names and numbers. Then create a string array specifying which columns from the Cursor you want in the layout for each result and an integer array specifying the corresponding views that each column should be placed:

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
   ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};
```

When you instantiate the SimpleCursorAdapter, pass the layout to use for each result, the Cursor containing the results, and these two arrays:

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
   R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);

ListView listView = findViewById(R.id.listView1);
listView.setAdapter(adapter);
```

The SimpleCursorAdapter creates a view for each row in the Cursor using the provided layout by inserting each from Columns item into the corresponding toViews view.

ListView and RecyclerView

RecyclerView:
The RecyclerView is a ViewGroup, more extensible as it provides both horizontal and vertical layouts. Use the RecyclerView widget when you have data collections whose elements change at runtime based on user action or network events.

Main RecyclerView Components:

RecyclerView.Adapter - Handles and binds data to the view
LayoutManager - Positions the items
ItemAnimator - Animates the items

Components of a RecyclerView:
A RecyclerView needs a layout manager and an adapter. A layout manager positions item views inside a RecyclerView and determines when to reuse item views no longer visible.

RecyclerView provides these built-in layout managers:

LinearLayoutManager shows items in a vertical or horizontal scrolling list.
GridLayoutManager shows items in a grid.
StaggeredGridLayoutManager shows items in a staggered grid.

RecyclerView.Adapter
RecyclerView includes a new kind of adapter. It requires ViewHolder class. You will have to override two main methods: one to inflate the view and its view holder, and another one to bind data to the view.

ItemAnimator:
RecyclerView.ItemAnimator will animate ViewGroup modifications such as add/delete/select that are notified to the adapter. DefaultItemAnimator can be used for basic default animations and works quite well. See the section of this guide for more information.

Using the RecyclerView:

Add RecyclerView library to the Gradle build
Define a model class to use as the data source
Add a RecyclerView to your activity to display the items
Create a custom row layout XML file to visualize the item
Create a RecyclerView.Adapter and ViewHolder to render the item
Bind the adapter to the data source to populate the RecyclerView

ListView and RecyclerView

1. Adding RecyclerView library:
Add RecyclerView library as a dependency in your app/build.gradle:

```
dependencies {
    ...
    implementation 'com.android.support:recyclerview-v7:28.0.0'
}
```

Click on "Sync Project with Gradle files" to let your IDE download the appropriate resources.

You can however, just drag and drop RecyclerView component on the layout design pane which will automatically add the dependency in the gradle.

2. Defining a Model
RecyclerView get data from a data source. For our example, we will use Manufacturers class which represents the data model being displayed by the RecyclerView:

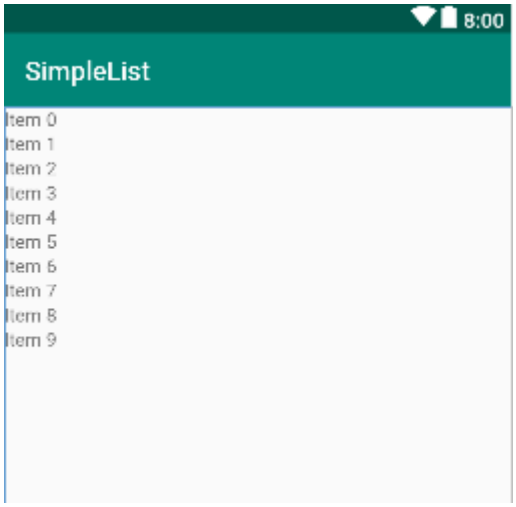
```
public class Manufacturer {
    private String mName;
    public Manufacturer(String name) {
        mName = name;
    }
    public String getName() {
        return mName;
    }
    public void setName(String name) {
        mName = name;
    }
}
```

3. Create the RecyclerView in the Activity Layout:
Add the RecyclerView in the activity layout XML file where you want the list to be displayed.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
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">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvManufacturer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
```

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

This results in the empty layout template as follows:



Now the RecyclerView is embedded within our activity layout file. Next, we can define the layout for each item within our list.

4. Creating the Custom Row Layout:
Define a layout file that will be used for each row within the list. This file contains a layout with a textview for the manufacturer name:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
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">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvManufacturer"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="0dp"
        android:layout_marginTop="38dp" />
</RelativeLayout>
```

5. Creating the RecyclerView.Adapter :

Create the adapter which will populate the data into the RecyclerView. The adapter's role is to convert an object at a position into a list row item to be inserted.

In RecyclerView the adapter requires a "ViewHolder" object which describes and provides access to all the views within each item row. We can create the basic empty adapter and holder together in ManufacturerRecyclerViewAdapter.java as follows:

```
// Note that we specify the custom ViewHolder which gives us access to our views
public class ManufacturerRecyclerViewAdapter
       extends
RecyclerView.Adapter<ManufacturerRecyclerViewAdapter.ViewHolder>{

      // Provide a direct reference to each of the views within a data item
    public class ViewHolder extends RecyclerView.ViewHolder
           implements View.OnClickListener {
       TextView myTextView;

      // We also create a constructor that accepts the entire item row
      // and does the view lookups to find each subview

      ViewHolder(View itemView) {
           super(itemView);
           myTextView = itemView.
                          findViewById(R.id.tvManufacturerName);
           itemView.setOnClickListener(this);

       }
}
```

Now we've defined the basic adapter and ViewHolder, we need to code up our adapter. Declare a member variable for the list of Manufacturers and get the list through constructor:

```
public class ManufacturerRecyclerViewAdapter
       extends
RecyclerView.Adapter<ManufacturerRecyclerViewAdapter.ViewHolder>{
    private List<Manufacturer> mData;
    private LayoutInflater mInflater;
    private ManufacturerRecyclerViewAdapter.ItemClickListener
mClickListener;

         // ... view holder defined above...

    // data is passed into the constructor
    ManufacturerRecyclerViewAdapter(Context context,
List<Manufacturer> data) {
       this.mInflater = LayoutInflater.from(context);
       this.mData = data;
    }
```

```
    // inflates the row layout from xml when needed
    @Override
    public ManufacturerRecyclerViewAdapter.ViewHolder
        onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = mInflater.inflate(R.layout.recyclerview_row,
parent, false);
        return new ManufacturerRecyclerViewAdapter.ViewHolder(view);
    }
    // binds the data to the TextView in each row
    @Override
    public void
onBindViewHolder(ManufacturerRecyclerViewAdapter.ViewHolder
holder,
                                int position) {
        Manufacturer manufacturer = mData.get(position);
        holder.myTextView.setText(manufacturer.getName());
    }
    // total number of rows
    @Override
    public int getItemCount() {
        return mData.size();
    }

    // convenience method for getting data at click position
    String getItem(int id) {
        return mData.get(id).getName();
    }
    // allows click events to be caught
    void
setClickListener( ManufacturerRecyclerViewAdapter.ItemClickListene
r itemClickListener) {
        this.mClickListener = itemClickListener;
    }
    // parent activity will implement this method to respond to
click events
    public interface ItemClickListener {
        void onItemClick(View view, int position);
    }
}
```

Every adapter has three primary methods: **onCreateViewHolder** to inflate the item layout and create the holder, **onBindViewHolder** to set the view attributes based on the data and **getItemCount** to determine the number of items. The code above implemented all three to finish the adapter. Furthermore, it also includes the ItemClickListener event listener.

With the adapter completed, bind the data from the adapter into the RecyclerView.

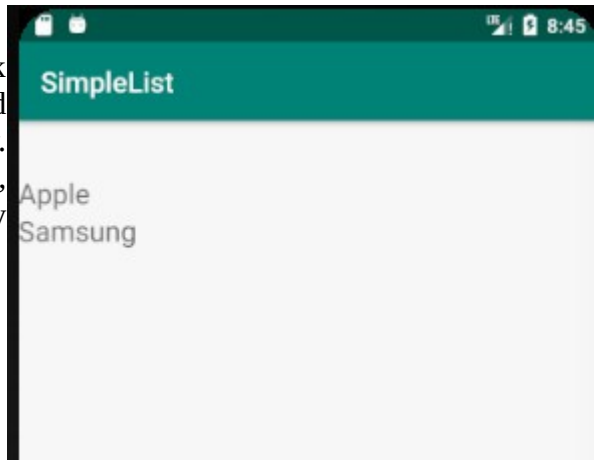
6. Binding the Adapter to the RecyclerView:

On our activity, we will populate sample manufacturers to be displayed in the RecyclerView.

```
public class MainActivity extends AppCompatActivity implements
ManufacturerRecyclerViewAdapter.ItemClickListener {
    private ManufacturerRecyclerViewAdapter adapter;
    private List<Manufacturer> mManufacturers = new ArrayList<>();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //MODEL
        mManufacturers.add( new Manufacturer("Apple"));
        mManufacturers.add( new Manufacturer("Samsung"));

        //setup the adapter
        adapter = new ManufacturerRecyclerViewAdapter(this,
mManufacturers);
        adapter.setClickListener(this);
        // set up the RecyclerView
        RecyclerView recyclerView =
findViewById(R.id.rvManufacturer);
        recyclerView.setLayoutManager(new
LinearLayoutManager(this));
        //recyclerView.setLayoutManager(new
GridLayoutManager(this, 2));
        //bind the adapter
        recyclerView.setAdapter(adapter);
    }
    @Override
    public void onItemClick(View view, int position) {
        Toast.makeText(this, "You clicked " +
adapter.getItem(position),      Toast.LENGTH_SHORT).show();
    }
}
```

Note that we have also implemented the item click listener event. Finally, compile and run the app and you should see something like the screenshot below. If you create enough items and scroll through the list, the views will be recycled and far smoother by default than the ListView widget.



ListView and RecyclerView

Some difference between ListView and RecyclerView:

ViewHolder - ListView adapters do not require the use of the ViewHolder. In contrast, implementing an adapter for RecyclerView requires the use of the ViewHolder for performance.
Customizable Layouts - ListView can only layout items in a vertical linear arrangement. RecyclerView has a RecyclerView.LayoutManager that allows any item layouts including horizontal lists or staggered grids.
Item Animations - ListView contains no special provisions to animate the addition or deletion of items. RecyclerView has the RecyclerView.ItemAnimator class for handling item animations.
Custom Adapter - ListView had adapters for different sources such as ArrayAdapter and CursorAdapter for arrays and database results respectively. RecyclerView.Adapter requires a custom implementation to supply the data to the adapter.
Item Decoration - ListView has android:divider property for dividers between items in list. RecyclerView uses RecyclerView.ItemDecoration object for more manual divider decorations.
Click Detection - ListView has a AdapterView.OnItemClickListener interface for binding to the click events for individual items in the list. RecyclerView only has support for RecyclerView.OnItemTouchListener to manage individual touch events.