Android Boot Camp

2

02 – Views and JSON

Ronald L. Ramos

August 2015

Topics

- CardViews & RecyclerViews
- Obtaining and Processing JSON data
- Using CardViews & RecyclerViews with JSON

PRESENTING DATA: CARDVIEWS & RECYCLER VIEWS

CardView and RecyclerView

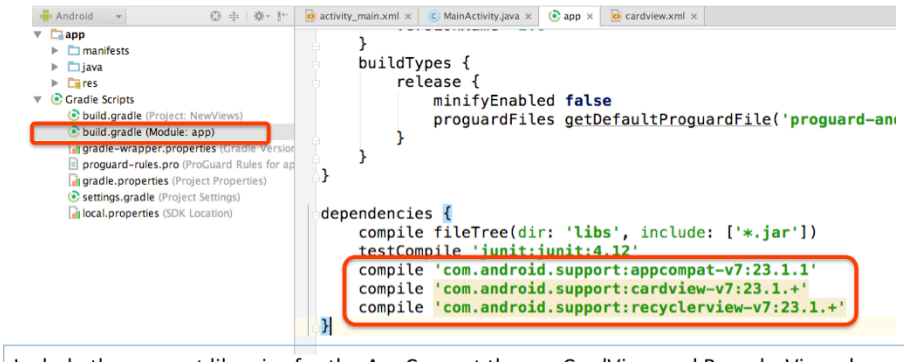
- Android's latest release, Lollipop (Android 5.0 API Level 21) includes the new RecyclerView and CardView widgets. They're also made available for use on devices running API Level 7 and above through the Support Libraries.
- The ***RecyclerView*** provides a more advanced and flexible way of displaying lists than the ListView.
- The ***CardView*** widget enables you to display views inside a card. You can design the card so that its look is consistent across your app.

http://www.101apps.co.za/index.php/articles/android-s-recyclerview-and-cardview-widgets.html

https://www.binpress.com/tutorial/android-l-recyclerview-and-cardview-tutorial/156

http://code.tutsplus.com/tutorials/getting-started-with-recyclerview-and-cardview-on-android--cms-23465

Setup before use



Include the support libraries for the AppCompat theme, CardView and RecyclerView classes in your build.gradle file

Our minimum SDK is 10 (Gingerbread_MR1) so we need to use the Support Libraries to:
Use the ActionBar
Use the CardView widget
Use the RecyclerView widget
In Android Studio, check your app's *build.gradle* file, making sure that it includes the appropriate dependencies

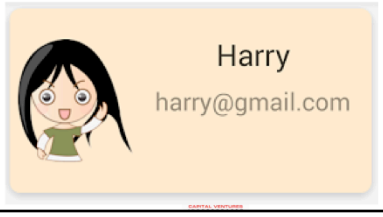
CODE:
compile **'com.android.support:appcompat-v7:23.1.1'**
compile **'com.android.support:cardview-v7:23.1.+'**
compile **'com.android.support:recyclerview-v7:23.1.+'**

NOTE: Make sure to change the number to reflect the latest SDK

THE CARDVIEW

CardView widgets

- Use the CardView widget to create cards that will look the same wherever you use them in your apps.
- You can specify their background, whether their corners should be rounded and if they should have a shadow.



An Empty CardView

```
<android.support.v7.widget.CardView
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

</android.support.v7.widget.CardView>
```

Sample CardView

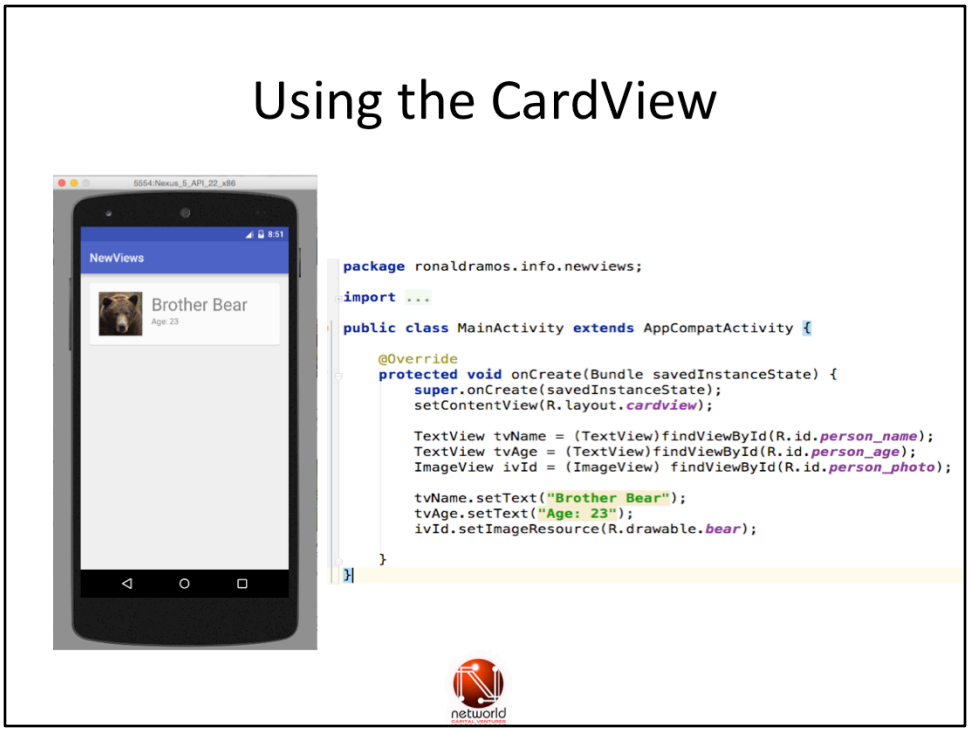
A more realistic CardView

- As a more realistic example, let us now create a LinearLayout and place a CardView inside it.
- The CardView could represent, for example, a person and contain the following:
  - a TextView to display the name of the person
  - a TextView to display the person's age
  - an ImageView to display the person's photo

We set the width of the card to match its parent so that the card stretches across the width of the screen. The height is set so that it stretches to enclose its biggest view, the *icon*.
We set the background colour using a colour resource. The *cardCornerRadius* attribute sets the radius of each of the card's corners.
The *cardElevation* attribute provides the shadow.
In addition, we include the text views and the image view within the CardView widget's tags (see the project code).

CODE:

```
<?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:padding="16dp"
   >

   <android.support.v7.widget.CardView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
```



CODE:

```
public class MainActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.cardview);

      TextView tvName = (TextView)findViewById(R.id.person_name);
      TextView tvAge = (TextView)findViewById(R.id.person_age);
      ImageView ivId = (ImageView) findViewById(R.id.person_photo);

      tvName.setText("Brother Bear");
      tvAge.setText("Age: 23");
      ivId.setImageResource(R.drawable.bear);

   }
}
```

THE RECYCLERVIEW

The RecyclerView Widget

- The RecyclerView widget is essentially a container that you can use to display large sets of data.
- It's very efficient as it only displays a few items at a time.
- **Views that are no longer needed are recycled and reused.** Not having to keep on inflating views saves CPU resources and valuable memory is saved by not having to keep views alive in the background.

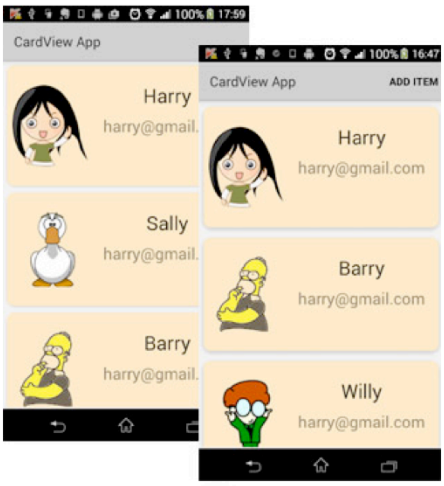
RecyclerView is more flexible that *ListView* even if it introduces some complexities. We all know how to use ListView in our app and we know if we want to increase the ListView performances we can use a pattern called *ViewHolder*. This pattern consists of a simple class that holds the references to the UI components for each row in the ListView. This pattern avoids looking up the UI components all the time the system shows a row in the list. Even if this pattern introduces some benefits, we can implement the ListView without using it at all. RecyclerView forces us to use the *ViewHolder pattern*.

Using the RecyclerView

- It's easy to use the RecyclerView Widget, simply:
  1. Specify an adapter – you need to create a custom adapter by extending the RecyclerView.Adapter class. The adapter will then display the data in the views that it creates
  2. Specify a layout manager – you'll need a layout manager to manage the positioning of items within the RecyclerView widget and also for recycling the views. There are three built-in layout managers:
     - LinearLayoutManager – where items appear in a vertical or horizontal scrolling list
     - GridLayoutManager - where items appear in a grid
     - StaggeredGridLayoutManager – where items appear in a staggered grid
- You can also create your own custom layout manager.
- You can animate the list items when they are added to or removed from the list. Adding and removing items are set by default to use the default animation. You can also use your own custom animations.

We display our list of items using the RecyclerView widget. Each item is displayed inside a CardView card.

A Sample app with CardView and RecyclerView



activity_main with RecyclerView

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="mat
    android:layout_height="match_parent" android:paddingLeft="@dimen/activity
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <android.support.v7.widget.RecyclerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/rv"
        />
</RelativeLayout>
```


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
   android:layout_height="match_parent" android:paddingLeft="@dimen/
activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">
   <android.support.v7.widget.RecyclerView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:id="@+id/rv"
      />
</RelativeLayout>
```

Data Structure: Person Object

```
package ronaldramos.info.newviews;

class Person {
    String name;
    String age;
    int photoId;

    Person(String name, String age, int photoId) {
        this.name = name;
        this.age = age;
        this.photoId = photoId;
    }
}
```

Just like a ListView, a RecyclerView needs an adapter to access its data. But before we create an adapter, let us create data that we can work with. Create a simple class to represent a person and then write a constructor method to initialize a Person object

CODE:

```
class Person {
   String name;
   String age;
   int photoId;

   Person(String name, String age, int photoId) {
      this.name = name;
      this.age = age;
      this.photoId = photoId;
   }
}
```

```
private List<Person> persons;

// This method creates an ArrayList that has three Person objects
// Checkout the project associated with this tutorial on Github if
```

Creating an Adapter

- To create an adapter that a RecyclerView can use, you must extend RecyclerView.Adapter. This adapter follows the **view holder** design pattern, which means that it you to define a custom class that extends RecyclerView.ViewHolder. This pattern minimizes the number of calls to the costly findViewById method.
- Earlier we already defined the XML layout for a CardView that represents a person. We are going to reuse that layout now. Inside the constructor of our custom ViewHolder, initialize the views that belong to the items of our RecyclerView.

Adapter: ViewHolder innerclass

```
public class RVAdapter extends RecyclerView.Adapter<RVAdapter.PersonViewHolder>{

    public static class PersonViewHolder extends RecyclerView.ViewHolder {
        CardView cv;
        TextView personName;
        TextView personAge;
        ImageView personPhoto;

        PersonViewHolder(View itemView) {
            super(itemView);
            cv = (CardView)itemView.findViewById(R.id.cv);
            personName = (TextView)itemView.findViewById(R.id.person_name);
            personAge = (TextView)itemView.findViewById(R.id.person_age);
            personPhoto = (ImageView)itemView.findViewById(R.id.person_photo);
        }
    }
}
```

CODE:

public class RVAdapter **extends**
RecyclerView.Adapter<RVAdapter.PersonViewHolder>{

public static class PersonViewHolder **extends** RecyclerView.ViewHolder {
CardView **cv**;
TextView **personName**;
TextView **personAge**;
ImageView **personPhoto**;

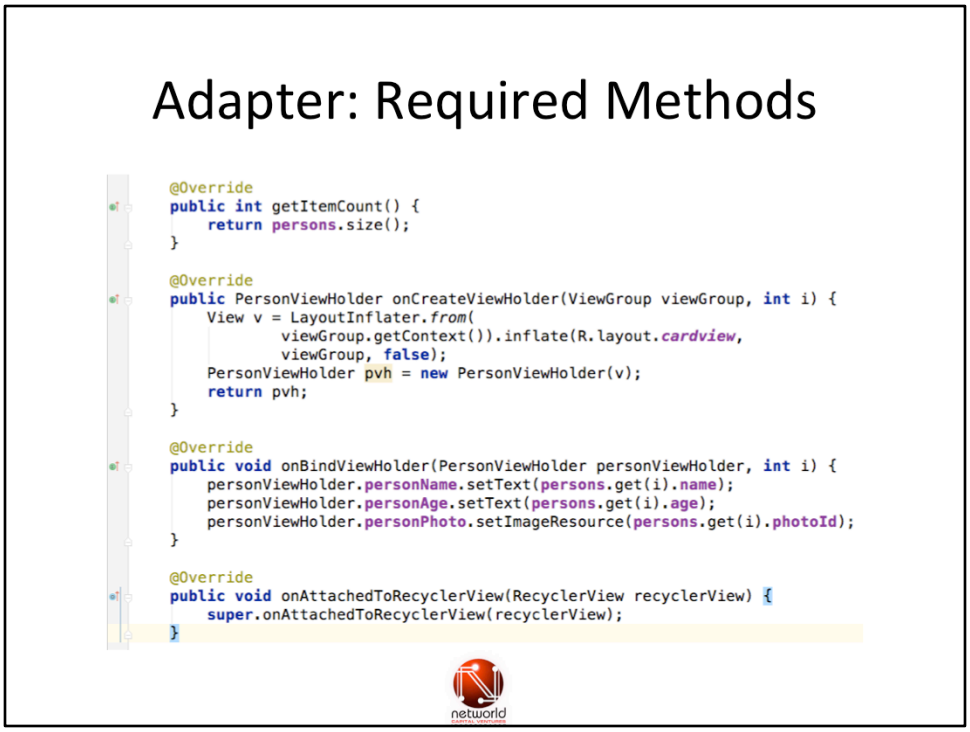
PersonViewHolder(View itemView) {
super(itemView);
cv = (CardView)itemView.findViewById(R.id.***cv***);
personName = (TextView)itemView.findViewById(R.id.***person_name***);
personAge = (TextView)itemView.findViewById(R.id.***person_age***);
personPhoto = (ImageView)itemView.findViewById(R.id.***person_photo***);
}
}

Adapter: Constructor & List of Persons

```
List<Person> persons;

RVAdapter(List<Person> persons){
    this.persons = persons;
}
```

CODE:

```
public class RVAdapter extends
RecyclerView.Adapter<RVAdapter.PersonViewHolder>{

   List<Person> persons;

   RVAdapter(List<Person> persons){
      this.persons = persons;
   }

   public static class PersonViewHolder extends RecyclerView.ViewHolder {
      CardView cv;
      TextView personName;
      TextView personAge;
      ImageView personPhoto;

      PersonViewHolder(View itemView) {
         super(itemView);
         cv = (CardView)itemView.findViewById(R.id.cv);
         personName = (TextView)itemView.findViewById(R.id.person_name);
```



RecyclerView.Adapter has three abstract methods that we must override.

Let us start with the getItemCount method. This should return the number of items present in the data. As our data is in the form of a List, we only need to call the size method on the List object.

Next, override the onCreateViewHolder method. As its name suggests, this method is called when the custom ViewHolder needs to be initialized. We specify the layout that each item of the RecyclerView should use. This is done by inflating the layout using LayoutInflater, passing the output to the constructor of the custom ViewHolder.

Override the onBindViewHolder to specify the contents of each item of the RecyclerView. This method is very similar to the getView method of a ListView's adapter. In our example, here's where you have to set the values of the name, age, and photo fields of the CardView.

Finally, you need to override the onAttachedToRecyclerView method. For now, we can simply use the superclass's implementation of this method as shown below.

CODE:

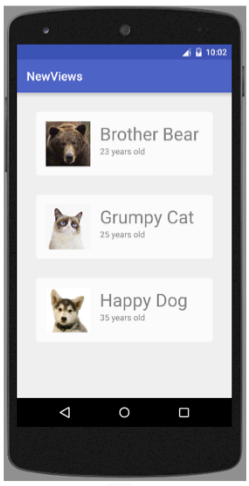
MainActivity

```
public class MainActivity extends AppCompatActivity {
    private List<Person> persons;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initializeData();
        RecyclerView rv = (RecyclerView)findViewById(R.id.rv);
        rv.setHasFixedSize(true);

        RVAdapter adapter = new RVAdapter(persons);
        rv.setAdapter(adapter);

        LinearLayoutManager llm = new LinearLayoutManager(this);
        rv.setLayoutManager(llm);
    }
    // This method creates an ArrayList that has three Person objects
    private void initializeData() {
        persons = new ArrayList<>();
        persons.add(new Person("Brother Bear", "23 years old", R.drawable.bear));
        persons.add(new Person("Grumpy Cat", "25 years old", R.drawable.cat));
        persons.add(new Person("Happy Dog", "35 years old", R.drawable.dog));
    }

}
```

CODE:

```
package ronaldramos.info.newviews;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
   private List<Person> persons;

   @Override
```

Running the App



A LITTLE BIT OF JSON ON OUR MINDS

What is JSON?

- JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.
- It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999.

JSON is very light weight, structured, easy to parse and much human readable. JSON is best alternative to XML when your android app needs to interchange data with your server

References:
httP://www.json.org/
http://www.androidhive.info/2012/01/android-json-parsing-tutorial/

Why JSON?

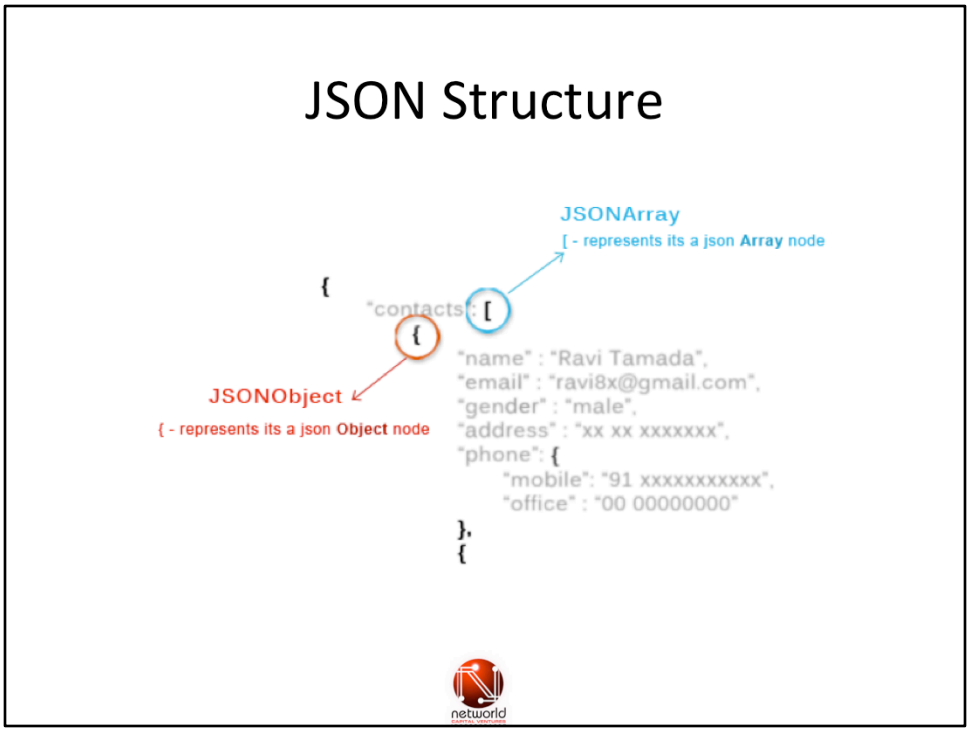
- You should use JSON in your projects instead of XML because:
  - it is lightweight
  - easier to parse
  - supported by most programming languages.

Sample JSON

```
{
    "contacts": [
        {
                "id": "c200",
                "name": "Ravi Tamada",
                "email": "ravi@gmail.com",
                "address": "xx-xx-xxxx,x - street, x - country",
                "gender" : "male",
                "phone": {
                    "mobile": "+91 0000000000",
                    "home": "00 000000",
                    "office": "00 000000"
                }
        },
        {
                "id": "c201",
                "name": "Johnny Depp",
                "email": "johnny_depp@gmail.com",
                "address": "xx-xx-xxxx,x - street, x - country",
                "gender" : "male",
                "phone": {
                    "mobile": "+91 0000000000",
                    "home": "00 000000",
                    "office": "00 000000"
                }
        },
        .
        .
        .
        .
    ]
}
```

Following is the sample JSON that we are going to parse in this tutorial. This is very simple JSON which gives us list of contacts where each node contains contact information like name, email, address, gender and phone numbers.

You can get this JSON data by accessing http://api.androidhive.info/contacts/



The difference between [and { – (Square brackets and Curly brackets)

In general all the JSON nodes will start with a square bracket or with a curly bracket. The difference between [and { is, the square bracket ([) represents starting of an JSONArray node whereas curly bracket ({) represents JSONObject. So while accessing these nodes we need to call appropriate method to access the data.

If your JSON node starts with [, then we should use getJSONArray() method. Same as if the node starts with {, then we should use getJSONObject() method.

HOW TO EXPOSE YOUR DATA

RESTful APIs

- ***RE**presentational **S**tate **T**ransfer*, an architectural style that can be used in building networked applications, is becoming increasingly popular nowadays.
- Many leading vendors have opened the doors of their services to developers, providing them with restful accesses to different web services.

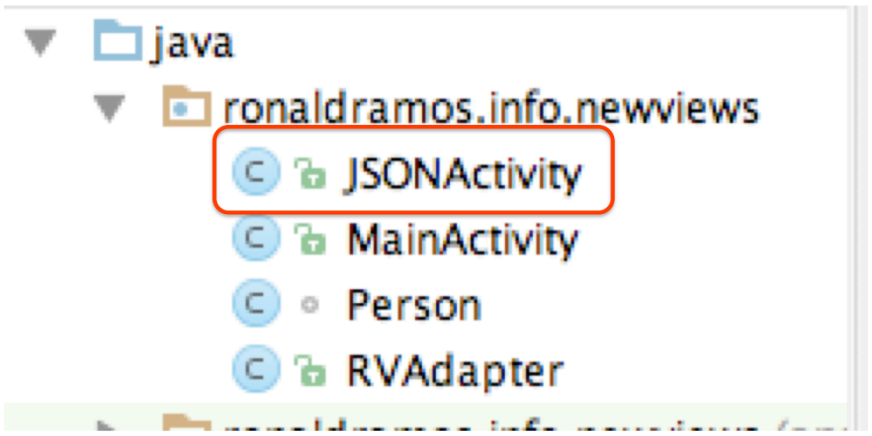
HTTP Methods to Facilitate REST

- GET to retrieve and search data
- POST to add data
- PUT to update data
- DELETE to delete data

https://docs.phalconphp.com/en/latest/reference/tutorial-rest.html

AN APP THAT ACCESSES JSON

Add a new BLANK Activity Class



We will reuse the CardView and RecyclerView that we created earlier

Modify AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ronaldramos.info.newviews" >
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="NewViews"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >

        </activity>
        <activity android:name=".JSONActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Add permission to access the internet

Make the new activity the launch activity

JSONActivity: Initializations

```
public class JSONActivity extends AppCompatActivity {
    private static final String TAG = "JSONExample";
    private List<Person> persons;
    private RecyclerView mRecyclerView;
    private RVAdapter adapter;
    private ProgressBar progressBar;
    private String urlString =
            "http://mobiledev.ronaldramos.info/getallcontacts.php";
```

We initialize some components we will use for processing data. We also initialize the source of the data.

CODE:

```
public class JSONActivity extends AppCompatActivity {
   private static final String TAG = "JSONExample";
   private List<Person> persons;
   private RecyclerView mRecyclerView;
   private RVAdapter adapter;
   private ProgressBar progressBar;
   private String urlString =
         "http://mobiledev.ronaldramos.info/getallcontacts.php";
```

JSONActivity: onCreate()

Activity_main is reused;
RecyclerView is reused

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mRecyclerView = (RecyclerView) findViewById(R.id.rv);
    mRecyclerView.setLayoutManager(new LinearLayoutManager(this));

    progressBar = (ProgressBar) findViewById(R.id.progressBar);
    progressBar.setVisibility(View.VISIBLE);

    new GetData().execute(urlString);
}
```

GetData inner AsyncTask class

- AsyncTask is an abstract class provided by Android which helps us to use the UI thread properly.
- This class allows us to perform long/ background operations and show its result on the UI thread without having to manipulate threads.

Why AsyncTask?

- Android implements a single thread model and whenever an android application is launched, a thread is created.
- Assuming we are doing network operation on a button click in our application. On button click a request would be made to the server and response will be awaited.
- Due to the single thread model of android, till the time response is complete our screen is non-responsive.
- So we should avoid performing long running operations on the main UI thread.
- This includes file and network access. To overcome this we can create new thread using an AsyncTask and implement run method to perform this network call, so that the UI remains responsive.

JSONActivity: GetData inner class

```
public class GetData extends
AsyncTask<String, Void, Integer>
{


}
```

AsyncTask Methods

- **doInBackground:** Code performing long running operation goes in this method. When onClick method is executed on click of button, it calls execute method which accepts parameters and automatically calls doInBackground method with the parameters passed.
- **onPostExecute:** This method is called after doInBackground method completes processing. Result from doInBackground is passed to this method.
- **onPreExecute:** This method is called before doInBackground method is called.
- **onProgressUpdate:** This method is invoked by calling publishProgress anytime from doInBackground call this method.

JSONActivity: GetData onPreExecute

```
@Override
protected void onPreExecute() {
    setProgressBarIndeterminateVisibility(true);
}
```

JSONActivity: GetData doInBackground()

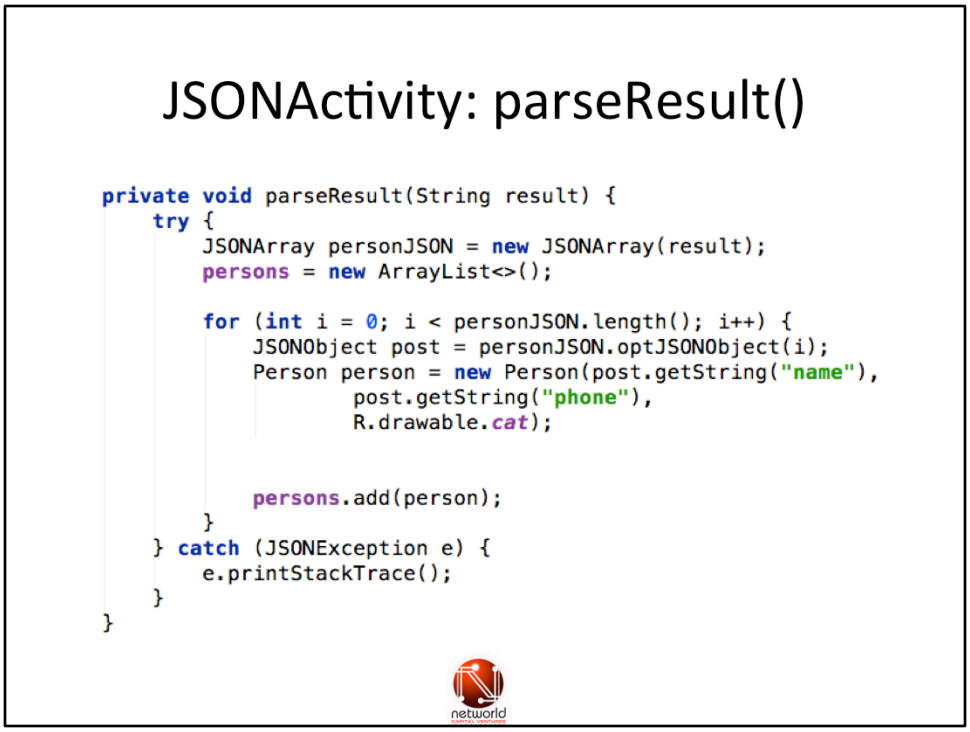
```
@Override
protected Integer doInBackground(String... params) {
    Integer result = 0;
    HttpURLConnection urlConnection;
    try {
        URL url = new URL(params[0]);
        urlConnection = (HttpURLConnection) url.openConnection();
        int statusCode = urlConnection.getResponseCode();

        // 200 represents HTTP OK
        if (statusCode == 200) {
            BufferedReader r = new BufferedReader(new InputStreamReader
                    (urlConnection.getInputStream()));
            StringBuilder response = new StringBuilder();
            String line;
            while ((line = r.readLine()) != null) {
                response.append(line);
            }
            parseResult(response.toString());
            result = 1; // Successful
        } else {
            result = 0; //"Failed to fetch data!";
        }
    } catch (Exception e) {
        Log.d(TAG, e.getLocalizedMessage());
    }
    return result; //"Failed to fetch data!";
}
```

JSONActivity: GetData onPostExecute()

```
protected void onPostExecute(Integer result) {
    // Download complete. Let us update UI
    progressBar.setVisibility(View.GONE);

    if (result == 1) {
        adapter = new RVAdapter(getApplicationContext(), persons);
        mRecyclerView.setAdapter(adapter);
    } else {
        Toast.makeText(getApplicationContext(),
                "Failed to fetch data!", Toast.LENGTH_SHORT).show();
    }
}
```

CODE:

```
public class GetData extends AsyncTask<String, Void, Integer> {
   @Override
   protected void onPreExecute() {
      setProgressBarIndeterminateVisibility(true);
   }

   @Override
   protected Integer doInBackground(String... params) {
      Integer result = 0;
      HttpURLConnection urlConnection;
      try {
         URL url = new URL(params[0]);
         urlConnection = (HttpURLConnection) url.openConnection();
         int statusCode = urlConnection.getResponseCode();

         // 200 represents HTTP OK
         if (statusCode == 200) {
            BufferedReader r = new BufferedReader(new InputStreamReader
```



parseResult() is a utility method to process JSON information and connect it to the CardView

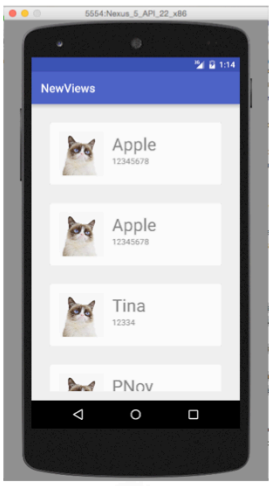
CODE:

```
private void parseResult(String result) {
   try {
      JSONArray personJSON = new JSONArray(result);
      persons = new ArrayList<>();

      for (int i = 0; i < personJSON.length(); i++) {
         JSONObject post = personJSON.optJSONObject(i);
         Person person = new Person(post.getString("name"),
               post.getString("phone"),
               R.drawable.cat);


         persons.add(person);
      }
   } catch (JSONException e) {
      e.printStackTrace();
```

Complete Code


```
package ronaldramos.info.newviews;

import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.Toast;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
```

Run the App


5554:Nexus_5_API_22_x86
1:14
NewViews
Apple
12345678
Apple
12345678
Tina
12334
PNov

END OF DAY 2