THỰC HÀNH LẬP TRÌNH DI ĐỘNG 2022

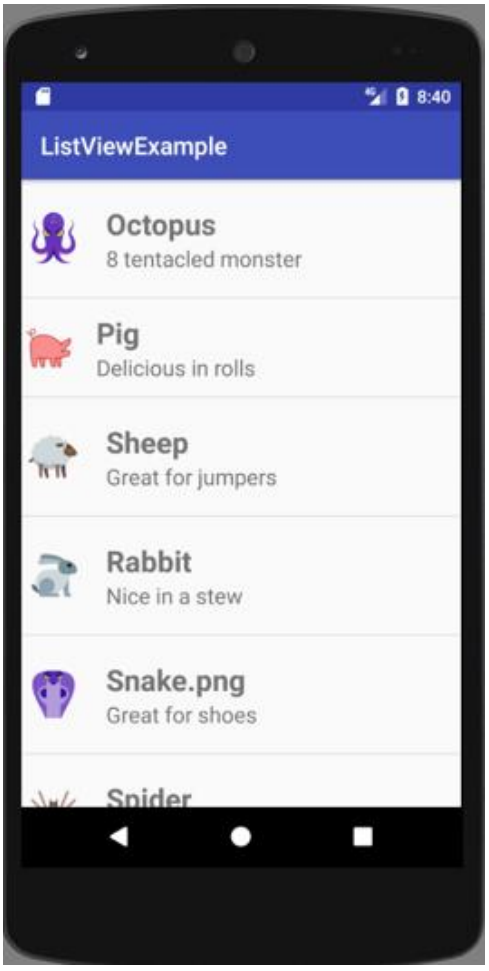
Lab 3: Listview

Yêu cầu:

- Thực hiện theo **nhóm thực hành 2-3 người** trong danh sách chia nhóm
- Nộp **project** có tên theo cú pháp *Lab3_NhomX* và file **video demo** cũng đặt tên theo cú pháp như trên, thêm **mô tả nhóm và thành viên nhóm** trong app

Nội dung:

- Xây dựng giao diện Activity với ListView như hình bên dưới

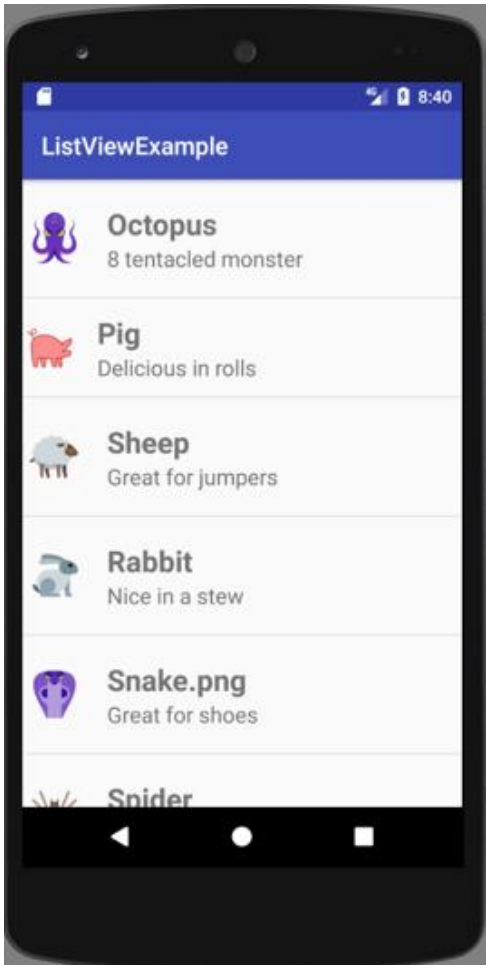


- Hoàn thiện giao diện cho Detail Activity theo layout tự chọn khi click vào đối tượng
- Thêm chức năng thêm, xóa, chỉnh sửa… đối tượng trong danh sách

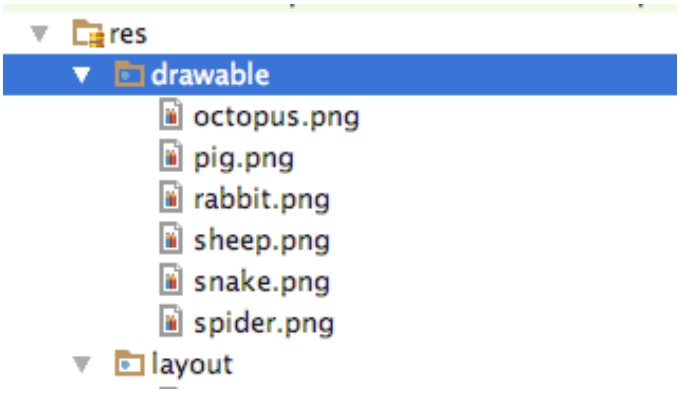
Gợi ý:

List Views in Android are one way to display a scrolling list of information—like a list of news items, a list of recipes, a list of delicious biscuits, whatever! They're one of the basic building blocks of a lot of apps and essential to understand if you want to make your own apps. This lab will show you step-by-step how to use them.

Design First: we need to create an listview application as below:



So before we do anything else, we'll copy and paste from Finder (Mac) or Windows Explorer, the images we'll be using as icons. I got mine from icons8.com as usual, size 96 by 96.



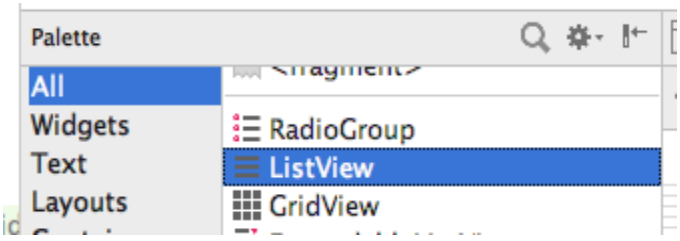
High-level steps

1. Add a **ListView** object to the layout file of an Activity
2. **Create a new XML file for the layout of the row in the ListView.** This is basically like a layout file for each row of the ListView, showing what and where goes in it.
3. **Create a "Custom Adapter" Java file.** This defines how your data (i.e. list of countries, dates, flag images etc) are mapped to the XML file
4. **Create your data—as arrays.** In the Activity's java file.
5. **Add code to create an instance of the Custom Adapter and populate with your data arrays.** In the Activity's java file.

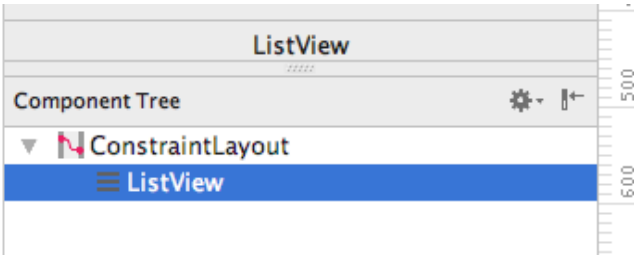
So a bit more convoluted than we've done so far, but it'll be worth it in the end!

Add a ListView

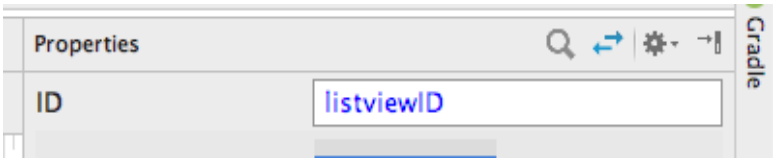
So, create an Empty app and go to the layout file. Delete the default TextView that's in there and add a **ListView**:



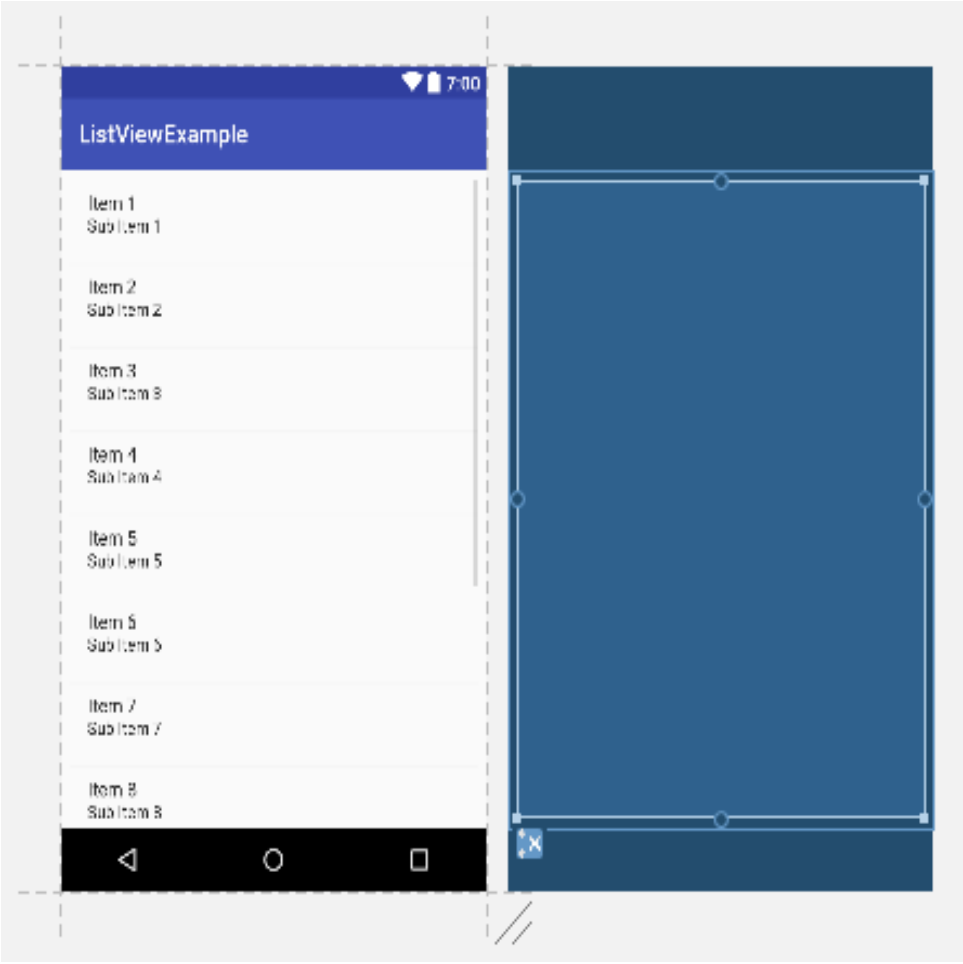
to the default **ConstraintLayout**:



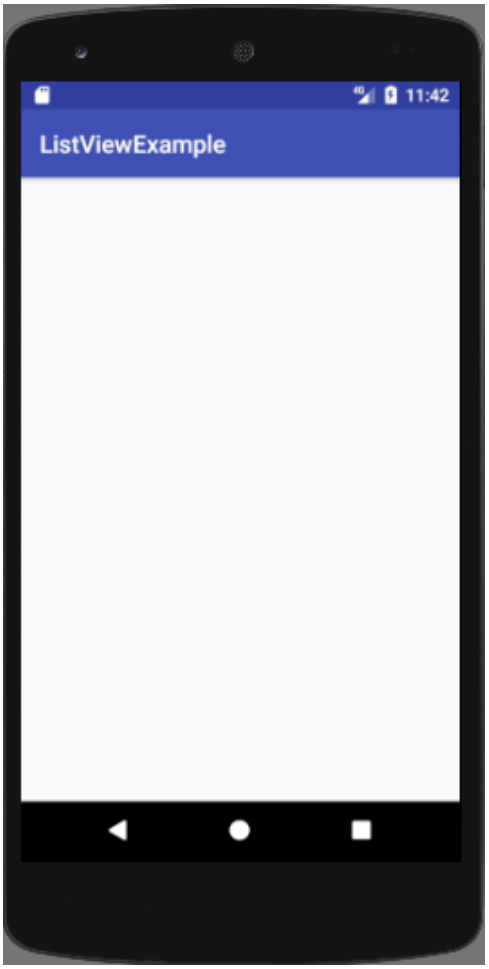
Give it an **ID**, like “listviewID”:



and the display should look like this:



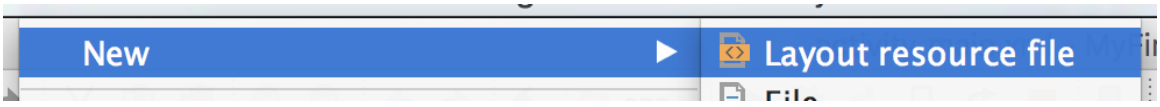
Note all the “Items” and “Subitems”—If you run the app, you’ll see these rows aren’t in the app—because we haven’t actually said what information / data should be displayed in them:



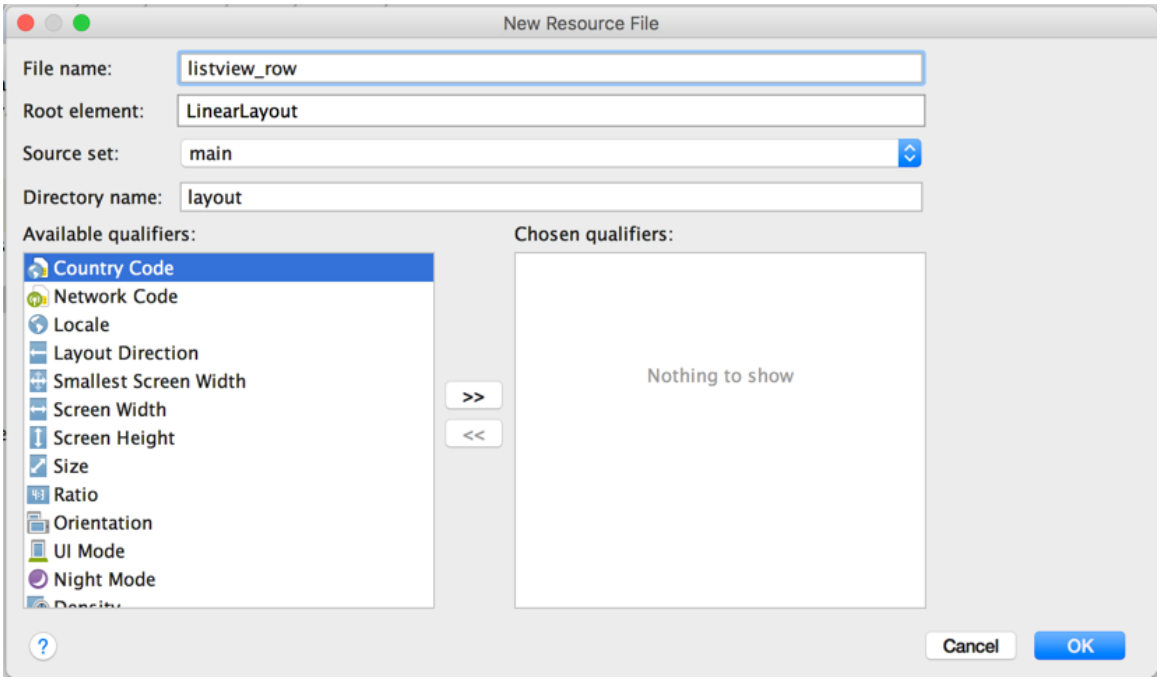
Before we define the data however, we're going to change the layout and appearance of these rows, to suit our design. We do this by using a new XML file that will define what the row looks like.

Create a new XML file for the row layout

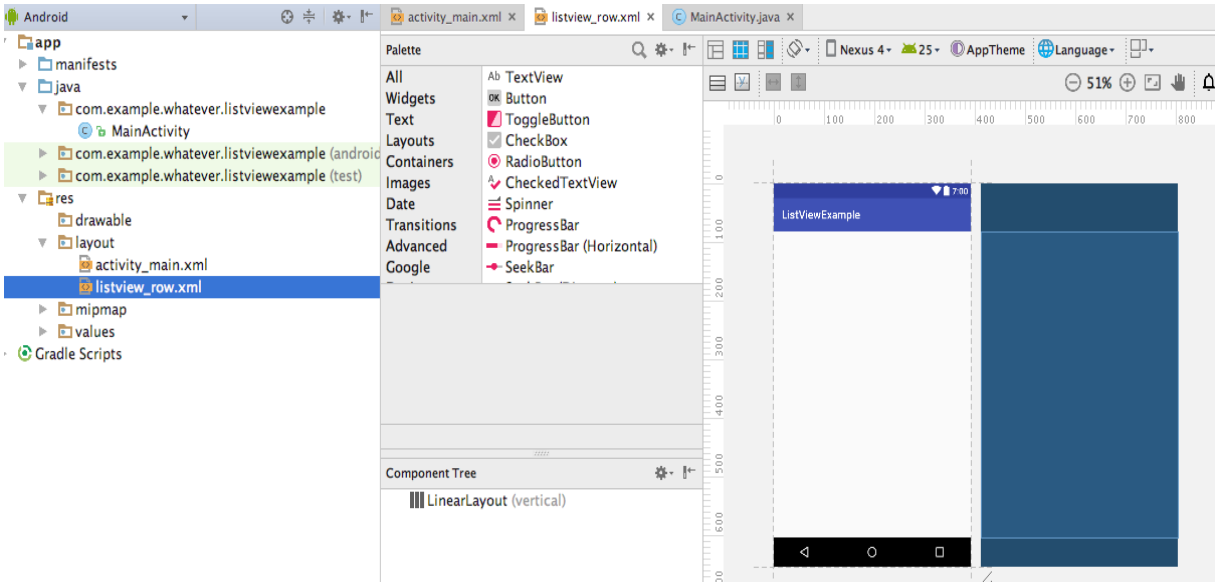
Right-click on the **res**->**layout** folder and select **New**->**Layout** resource file:



Give it a **File name**, here I use "listview_row":

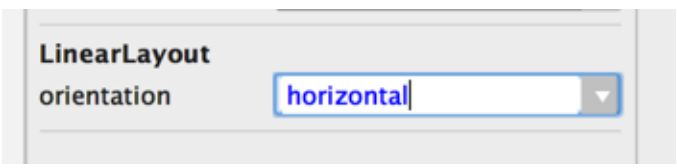


and hit **OK**. You will then have a new file called listview_row.xml in your layout folder:



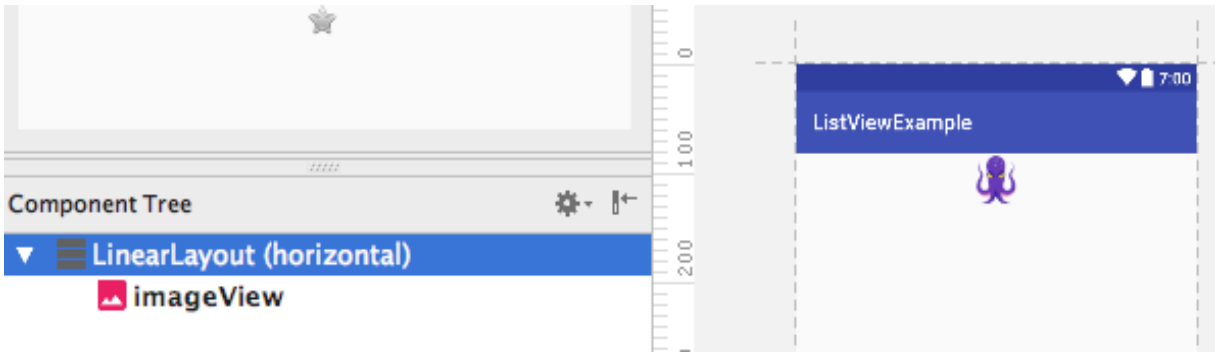
This will be used to show how each row of the ListView is laid out / designed. Each will have the same design—so we just define the layout once and then we write code to populate data into the rows.

First, change the **orientation** property of the **LinearLayout** to **horizontal**, to suit our design:

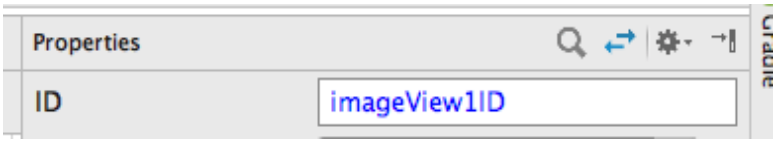


Then set the **layout_height** to **wrap_content**, so the row will only ever be as high as what's inside it:

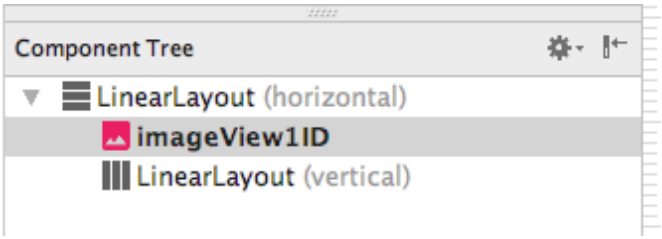
We leave the **layout_width** to **match_parent** as we want it as wide as the screen.
Now add an **ImageView** to the LinearLayout, and set its image to one of the animal icons we added earlier. Note that this image itself may not necessarily be used in the ListView, it's just used here as a placeholder.



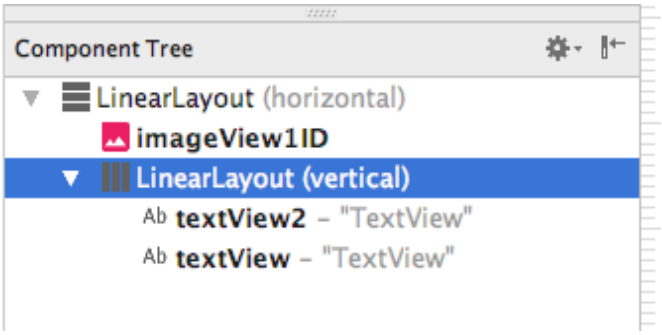
Give the imageView a new ID, I choose **imageView1ID:**



Now add a vertical **LinearLayout:**

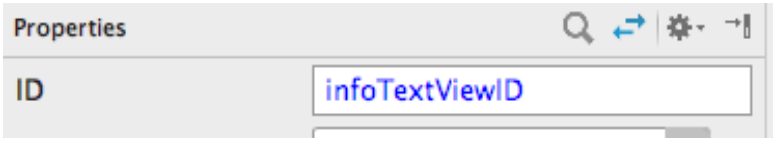


To this, we're going to add two **TextViews** to the vertical LinearLayout, so the Component Tree looks like this:

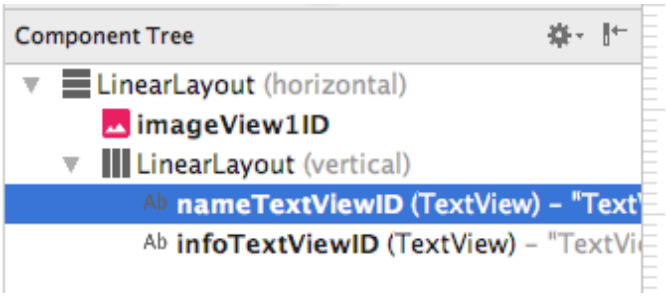


Change the IDs of the TextViews to nameTextViewID:

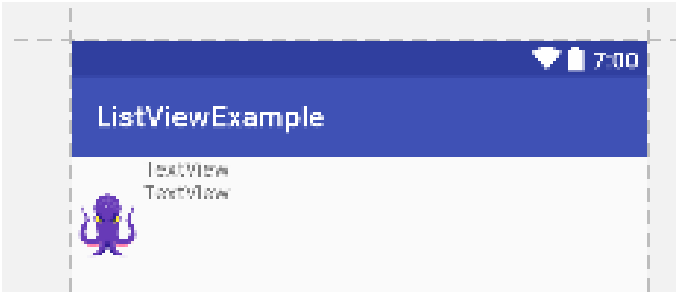
and infoTextViewID:



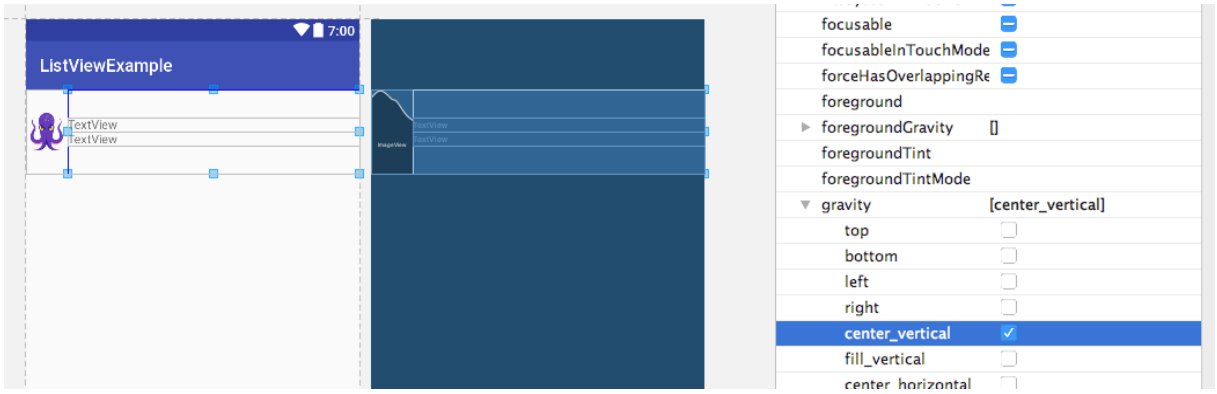
which will change the ComponentTree to look like this:



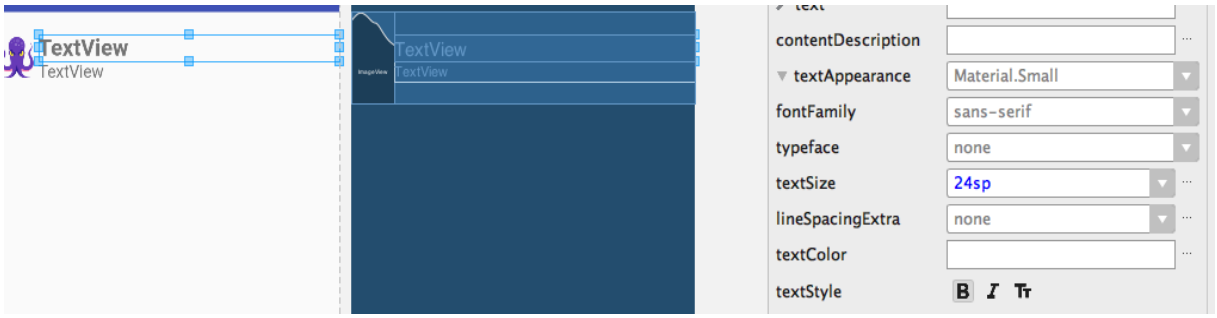
and the layout will look like this:



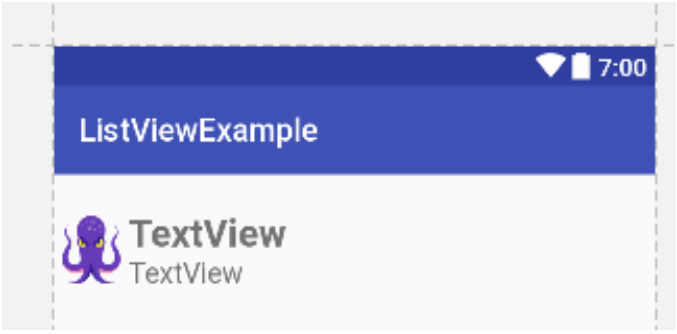
Let's change that a bit. First click on the LinearLayout and then in the extended properties, open out the **gravity** property and select **center_vertical**. This will move the two TextViews to the center of the row, vertically:



We then set the **textSize** of the first TextView to **24sp** and set the text to Bold in **textStyle**:



Then we set the **textSize** of the second TextView to 18sp, so the display looks like this:



Again—this layout file is for the row in the ListView. It's a template to specify how it will be laid out. The image shown is just a placeholder; we will be defining what actual images are shown later on.

For info, take a look at the XML code for the row layout, it should look something like this:

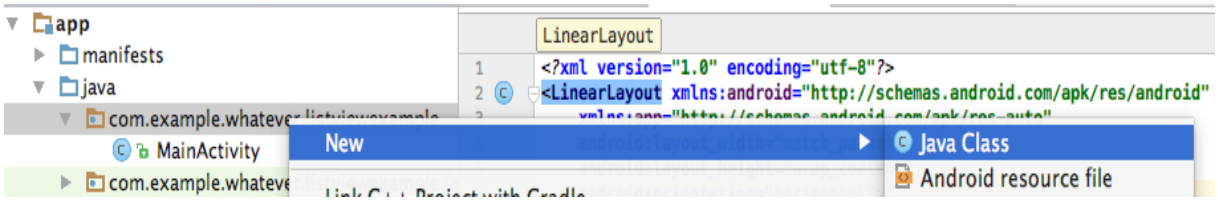
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
```

```
    <ImageView
        android:id="@+id/imageView1ID"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        app:srcCompat="@drawable/octopus" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center_vertical"
        android:layout_weight="1"
        android:gravity="center_vertical"
        android:orientation="vertical">

        <TextView
            android:id="@+id/nameTextViewID"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="TextView"
            android:textSize="24sp"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/infoTextViewID"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="TextView"
            android:textSize="18sp" />
    </LinearLayout>
</LinearLayout>
```

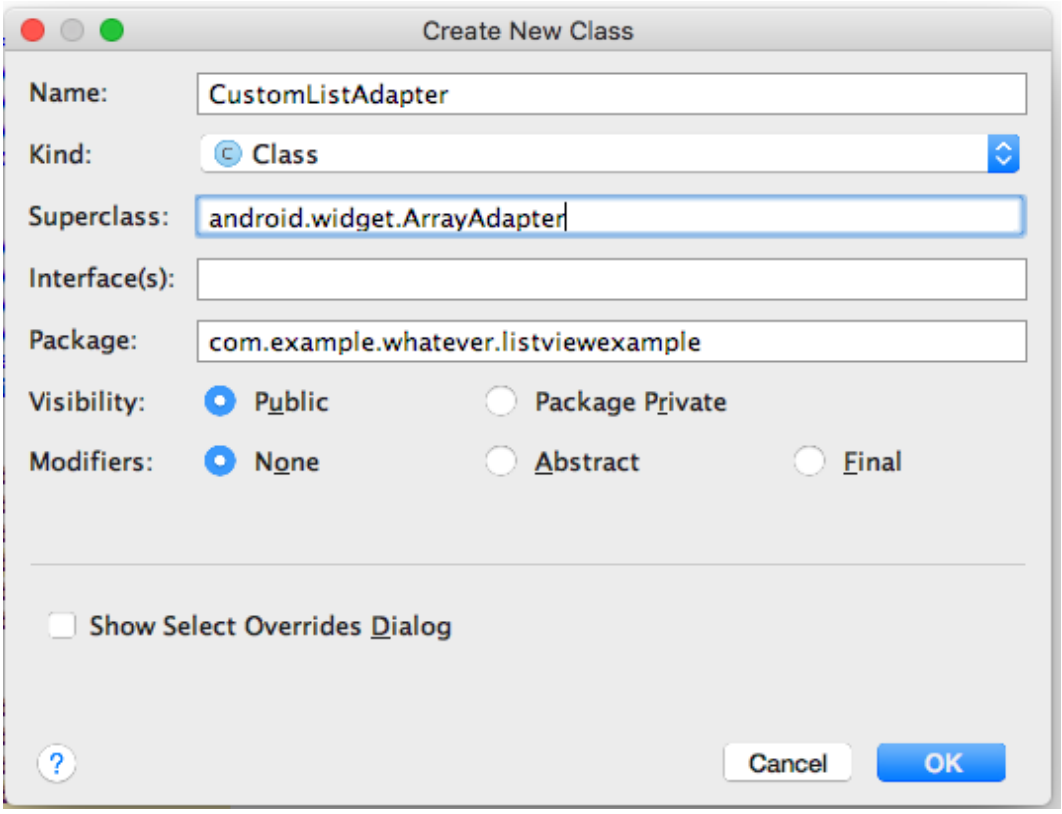
Custom Adapter Class

Ok, now we need to add a **Customer Adapter Class**. This is basically a class that sits between our main Java file and our row layout file—we use it to populate the rows of the ListView with data from the Java file.

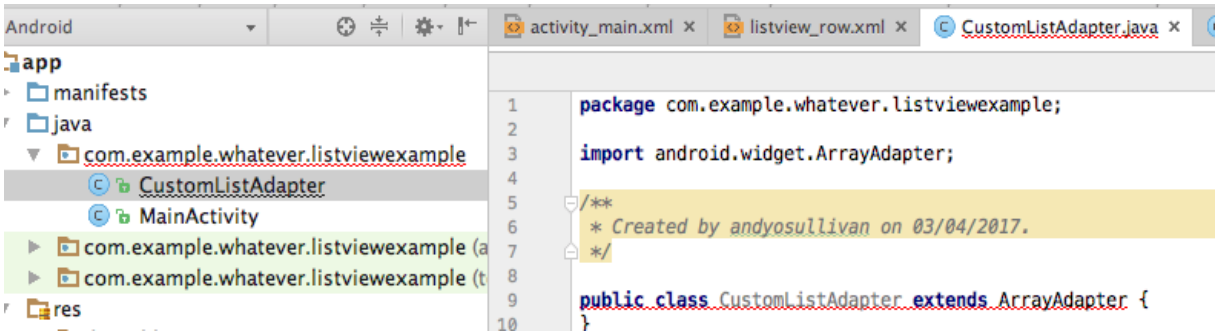
So, to add a new Class, right click on the folder where the rest of the java files are and select New-> "Java Class" file:



On the next window give it a name like "CustomListAdapter" and set the **Superclass** to "android.widget.ArrayAdapter":



Hit **OK** and it'll create this file:



This file **extends** the **ArrayAdapter** Class. This means roughly that it has the same attributes / functions as the ArrayAdapter Class. It's got those red squiggly error lines at the minute as we haven't implemented one of those functions yet, we'll do that in a moment. First, lets add some new **variables**that are properties of this Class:

public class CustomListAdapter **extends** ArrayAdapter {

//to reference the Activity
private final Activity **context**;

//to store the animal images
private final Integer[] **imageIDarray**;

//to store the list of countries
private final String[] **nameArray**;

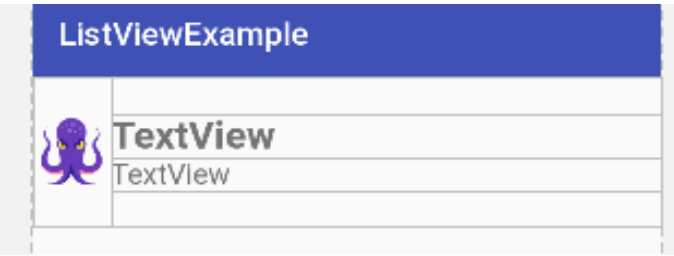
//to store the list of countries
private final String[] **infoArray**;

Remember that **Classes** can have **properties**—values specific to them e.g. a **Car** Class may have a numberOfDoors property or an engineSize property. Here, our **CustomListAdapter** Class has 4 properties. The first:

private final Activity **context**;

will be used to store what Activity the ListView is on and the three others—"nameArray", "imageIDArray" and "infoArray" will store the actual data that will be shown on screen, for

the names, information and images on animals. Make sense? These match the three fields from our **listview_row.xml** design:



If we had four other fields in there, we'd need four more properties in our CustomListAdapter class. Ok, let's now add a method:

```
public CustomListAdapter(Activity context, String[] nameArrayParam, String[]
infoArrayParam, Integer[] imageIDArrayParam){
   super(context,R.layout.listview_row , nameArrayParam);
}
```

This is known as a **constructor** method—a method which is used to create an instance of this Class. (We will be creating an instance of the this Class shortly!). It takes in as **parameters**:

- contentParam: which is an Activity
- nameArrayParam: which is an Array of Strings
- infoArrayParam: which is an Array of Strings
- imageIDArrayParam: which is an Array of Integers, which are the IDs of images

This basically means that we can create an instance of this Class and give it three arrays of data; that will be eventually shown on screen in the ListView. I added the "Param" at the end of each just to reenforce what they are; you could call them anything you wanted.

And note again— the three parameters "nameArrayParam", "infoArrayParam" and "imageIDArrayParam" are there as they match the design we have in **listview_row.xml**. If we had a different design, we would have different parameters.

This line:

```
super(context,R.layout.listview_row , nameArrayParam);
```

references R.layout.***listview_row***—this is the XML file we created to define what the row looks like. If you called yours mickey_mouse.xml, then here you'd write:

super(context,R.layout.***mickey_mouse*** , nameArrayParam);

The “nameArrayParam” is one of the the String[] parameters. **This is needed—it won’t work without it**! Just put it in!
Ok, let’s add more code to the constructor method:

```
public CustomListAdapter(Activity context, String[] nameArrayParam, String[]
infoArrayParam, Integer[] imageIDArrayParam){
   super(context,R.layout.listview_row , nameArrayParam);
   this.context=context;
   this.imageIDarray = imageIDArrayParam;
   this.nameArray = nameArrayParam;
   this.infoArray = infoArrayParam;
}
```

Those new 4 lines just store the passed in **parameters** into the corresponding **properties**.
Ok, let’s add one more method, called **getView**. This method is used automatically by the app to populate the data into each row.

```
public View getView(int position, View view, ViewGroup parent) {
   LayoutInflater inflater=context.getLayoutInflater();
   View rowView=inflater.inflate(R.layout.listview_row, null,true);
   //this code gets references to objects in the listview_row.xml file
   TextView nameTextField = (TextView) rowView.findViewById(R.id.nameTextViewID);
   TextView infoTextField = (TextView) rowView.findViewById(R.id.infoTextViewID);
   ImageView imageView = (ImageView) rowView.findViewById(R.id.imageView1ID);
   //this code sets the values of the objects to values from the arrays
   nameTextField.setText(nameArray[position]);
   infoTextField.setText(infoArray[position]);
   imageView.setImageResource(imageIDarray[position]);
   return rowView;
};
```

Bear with me! This **getView** method basically maps the data from the **properties** (nameArray etc) to the write fields in the **listview_row.xml**design. This line:

View rowView=inflater.inflate(R.layout.***listview_row***, **null**,**true**);

mentions “listview_row” so if you called your row xml file anything else, change it here. These lines:

```
TextView nameTextField = (TextView) rowView.findViewById(R.id.nameTextViewID);
TextView infoTextField = (TextView) rowView.findViewById(R.id.infoTextViewID);
ImageView imageView = (ImageView) rowView.findViewById(R.id.imageView1ID);
```

create variables from the XML file, just like we always do, using **findViewById** and the IDs we gave the fields earlier. These lines:

```
//this code sets the values of the objects to values from the arrays
nameTextField.setText(nameArray[position]);
infoTextField.setText(infoArray[position]);
imageView.setImageResource(imageIDarray[position]);
```

then set the fields to the values in the arrays. The **position** is a parameter the getView method has by default, it’s the index of which row is currently being filled out.
Ok, that’s the CustomListAdapter done. Probably a bit confusing but try it yourself, slowly!

Add the actual data

Now, we add the actual data to be shown on screen. We do this in the **MainActivity** Java file. After the class declaration:

```
public class MainActivity extends AppCompatActivity {
```

add these three arrays:

```
String[] nameArray = {"Octopus","Pig","Sheep","Rabbit","Snake","Spider" };

String[] infoArray = {
```

```
        "8 tentacled monster",
        "Delicious in rolls",
        "Great for jumpers",
        "Nice in a stew",
        "Great for shoes",
        "Scary."
};

Integer[] imageArray = {R.drawable.octopus,
        R.drawable.pig,
        R.drawable.sheep,
        R.drawable.rabbit,
        R.drawable.snake,
        R.drawable.spider};
```

The **nameArray** stores the names of the animals, the **infoArray** stores interesting facts about them and the **imageArray** stores the images. Note the syntax used for the images ***R.drawable.pig*** i.e. *R.drawable.<name of image>* without ".png" or ".jpg" at the end.

Also add this variable, for the **ListView** itself:

```
ListView listView;
```

Now, we create and instance our CustomListAdapter and feed these data arrays into it, then tell the ListView to use it. In the **onCreate** method, add these lines:

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   CustomListAdapter whatever = new CustomListAdapter(this, nameArray, infoArray,
imageArray);
   listView = (ListView) findViewById(R.id.listviewID);
   listView.setAdapter(whatever);
}
```

This line:

CustomListAdapter whatever = **new** CustomListAdapter(**this**, **nameArray**, **infoArray**, **imageArray**);

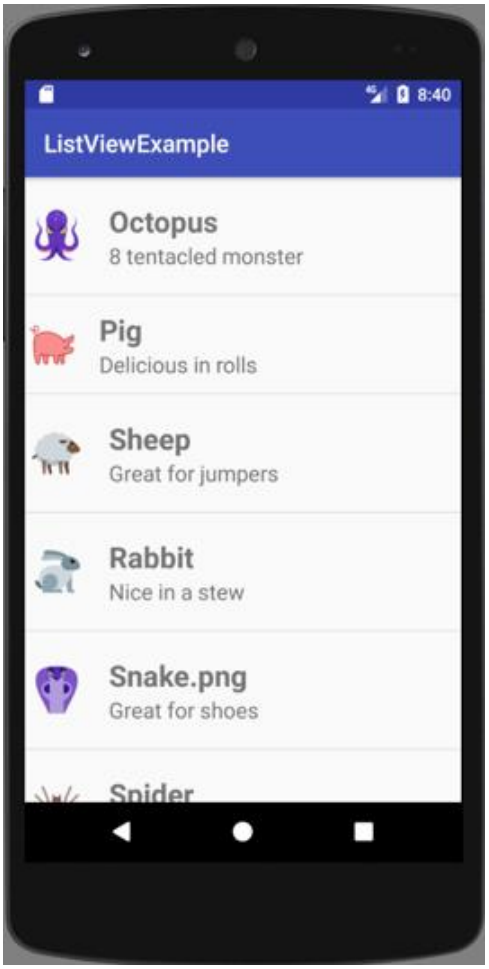
creates an instance of our CustomListAdapter class and gives the data arrays we just created as its parameters.

listView = (ListView) findViewById(R.id.***listviewID***);

gets a reference to the actual ListView object from the **activity_main.xml**layout file and

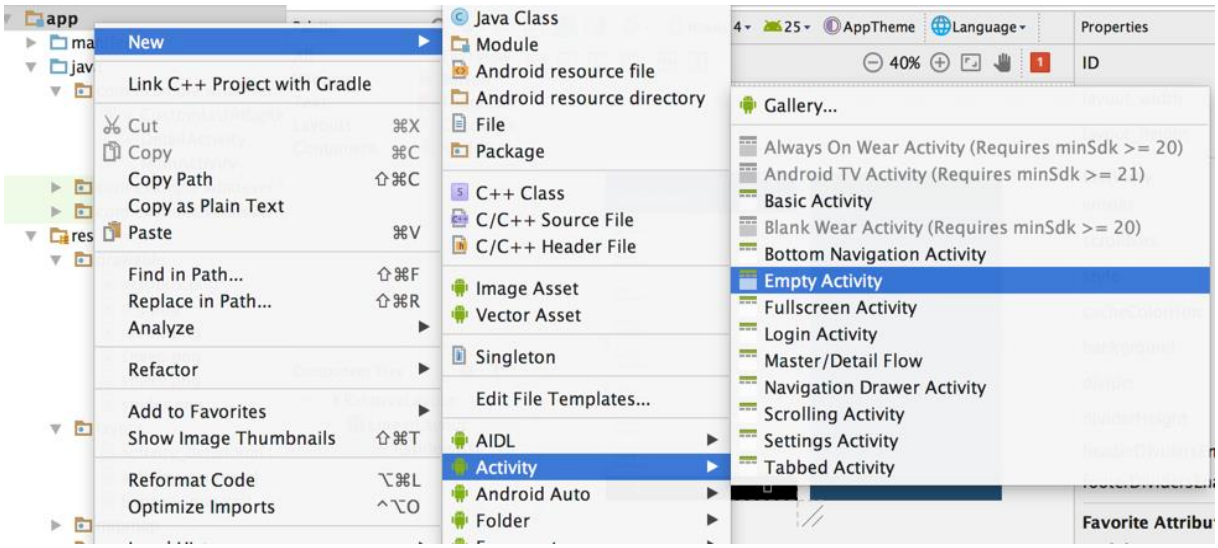
listView.setAdapter(whatever);

tells the ListView to use our adapter. Ok, run the app!

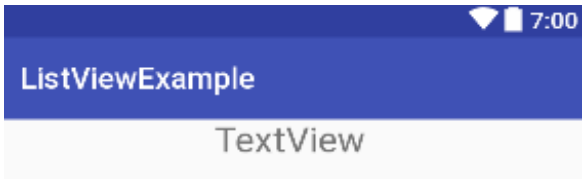


Awesome! Ok, one more thing. We want to do something when the user clicks on a row - go to a new page with details on that animal.

So - add a new Activity and call it something like Detail Activity:



In the **activity_detail.xml**, I just add a TextView, centre it on screen:



and give it an **ID** of "textID":

so we can set its text via code as usual!

Go back to **MainActivity** and add this code to **onCreate**:

```
listView.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
                            long id) {
    }
});
```

This is adding a “ItemClickListener” to the ListView. This basically “listens” for clicks on rows in the ListView, and provides the **position**—this is which row was clicked, with the first row being position 0, the second being position 1 and so on.
If it isn’t there already, add this at the top of the file, with the other imports:

```
import android.widget.AdapterView.OnItemClickListener;
```

Now, add these lines in bold to the onCreate method:

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

   CustomListAdapter whatever = new CustomListAdapter(this, nameArray, infoArray,
imageArray);
   listView = (ListView) findViewById(R.id.listviewID);
   listView.setAdapter(whatever);

   listView.setOnItemClickListener(new OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> parent, View view, int position,
                        long id) {
         Intent intent = new Intent(MainActivity.this, DetailActivity.class);
         String message = nameArray[position];
         intent.putExtra("animal", message);
         startActivity(intent);
      }
   });
}
```

The first creates an **intent**, which as we know is what we use to go to another Activity:

```
Intent intent = new Intent(MainActivity.this, DetailActivity.class);
```

This creates a new String variable and stores in it the value from the nameArray at the index given by position i.e. what row was clicked:

String message = nameArray[position];

This adds that to the intent as an Extra:

intent.putExtra("animal", message);

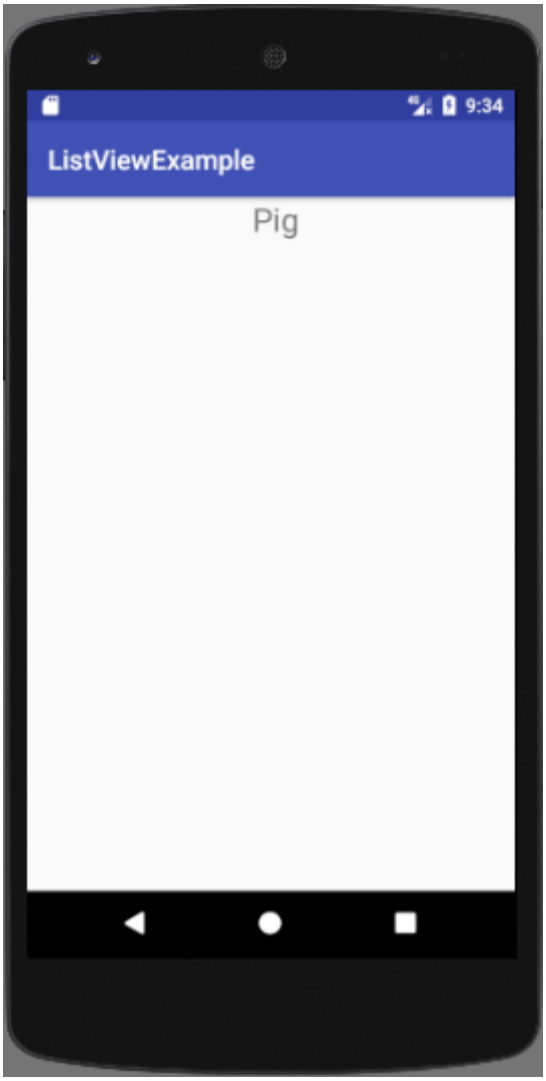
and this then starts the new Activity (going to the Detail page) using the intent:

startActivity(intent);

Now, go to **DetailActivity.java** and in **onCreate** add this:

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

    String savedExtra = getIntent().getStringExtra("animal");
    TextView myText = (TextView) findViewById(R.id.textID);
    myText.setText(savedExtra);
}
```

This gets the Extra value and displays it in the TextView. Run the app and hit one of the row, you should see:



Ok, we're done! Much more complex than most stuff we've done so far, but once you figure out the different steps:

- Adding a ListView
- Creating a new XML file for the layout of the ListView rows
- Creating a new custom Adapter class which maps data from arrays to the ListView rows
- Creating an instance of the Adapter class in our main java file and using it with the ListView