USER INTERFACES

COMP 41690

DAVID COYLE > D.COYLE@UCD.IE

UI OVERVIEW

All user interface elements in an Android app are built using **View** and **ViewGroup** objects.

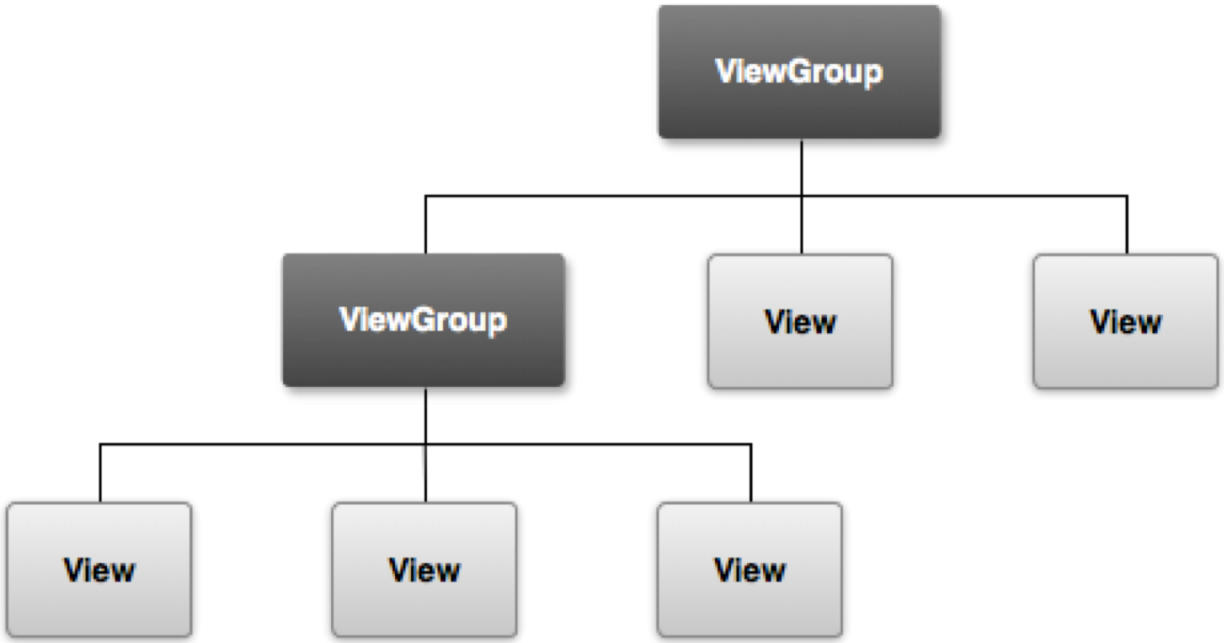
- A View is an object that draws something on the screen that the user can interact with, e.g. input controls or other widgets.

- A ViewGroup is an invisible object that holds other View (and ViewGroup) objects in order to define the layout of the interface.

Android provides a collection of both View and ViewGroup subclasses that offer you common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

UI OVERVIEW

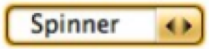
The user interface for each component of your app is defined using a hierarchy of View and ViewGroup objects.



This hierarchy tree can be as simple or complex as you need it to be (but simplicity is best for performance).

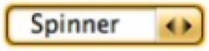
EXAMPLE VIEWS

There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other content.

 Analog/DigitalClock	 Button	 Checkbox	 Date/TimePicker
 EditText	 Gallery	 ImageView/Button	 ProgressBar
 RadioButton	 Spinner	 TextView	 MapView, WebView

EXAMPLE VIEWS

There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other content.

 Analog/DigitalClock	 Button	 Checkbox	 Date/TimePicker
 EditText	 Gallery	 ImageView/Button	 ProgressBar
 RadioButton	 Spinner	 TextView	 MapView, WebView

These are inbuilt ViewGroup objects. Each contains multiple View objects

LAYOUTS

Android provides a range of ViewGroup classes that provide standard ways to display the views nested within it.

Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

Displays web pages.

LAYOUT RESOURCE FILES

Each layout file must contain exactly one root element, which must be a View or ViewGroup object.

Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.

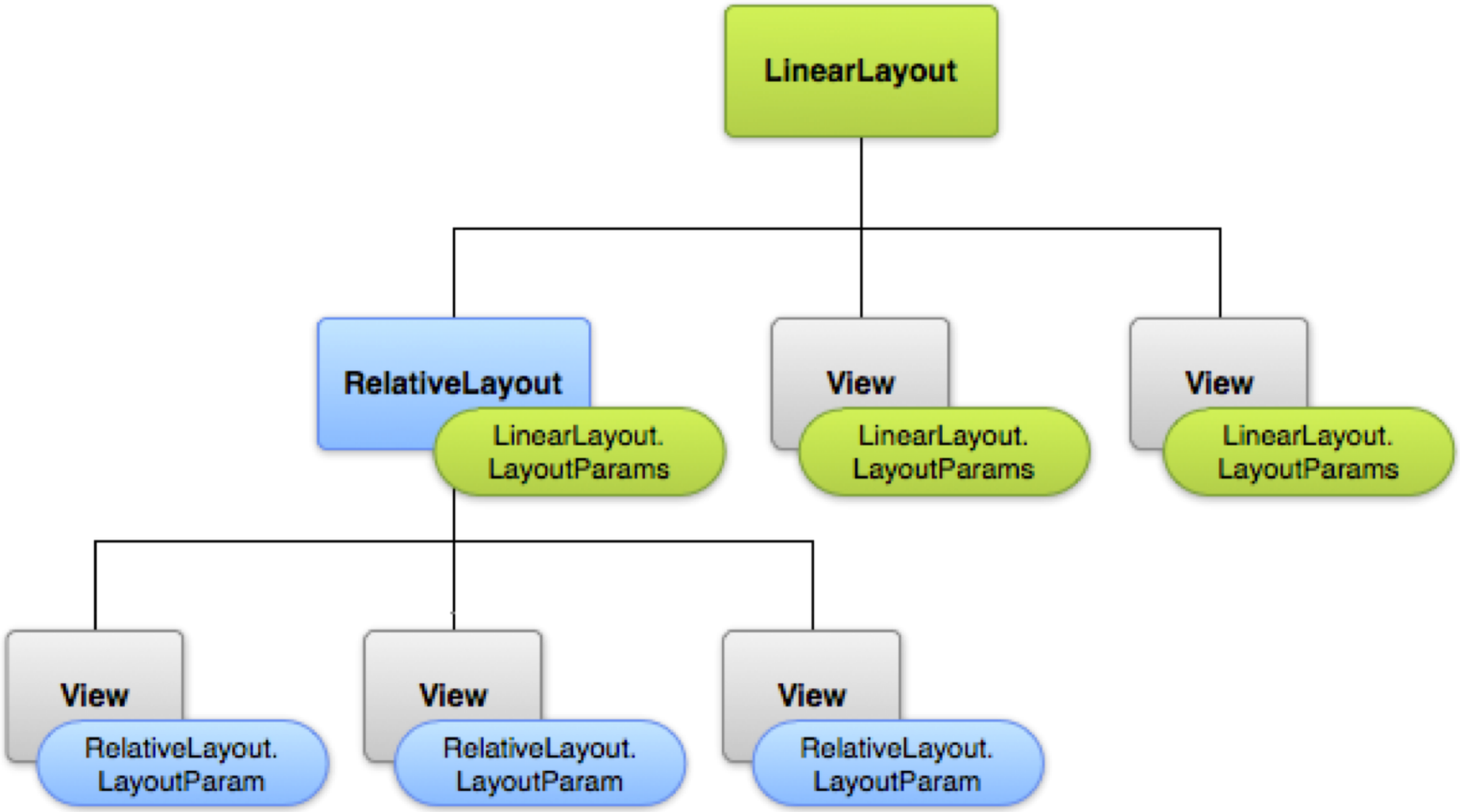
After you've declared your layout in XML, save the file with the .xml extension, in your Android project's res/layout/ directory, so it will properly compile.

```
<?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:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

XML layout attributes named layout_something define layout parameters for the View that are appropriate for the ViewGroup in which it resides.

LAYOUT PARAMETERS

XML layout attributes named layout_something define layout parameters for the View that are appropriate for the ViewGroup in which it resides.



LOADING A RESOURCE

When you compile your application, each XML layout file is compiled into a View resource.

You should load the layout resource from your application code, in your Activity.onCreate() callback implementation.

Do so by calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.layout_file_name.

E.g. if your XML layout is saved as main_layout.xml, you would load it for your Activity like so:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

VIEW CLASS

There are typically a few types of common operations you may wish to perform:

Set properties: for example setting the text of a TextView.

The available properties and the methods that set them will vary among the different subclasses of views. Note that properties that are known at build time can be set in the XML layout files.

Set focus: The framework will handled moving focus in response to user input. To force focus to a specific view, call requestFocus().

Set up listeners: Views allow clients to set listeners that will be notified when something interesting happens to the view.

For example, all views will let you set a listener to be notified when the view gains or loses focus. Similarly a Button exposes a listener to notify clients when the button is clicked.

Set visibility: You can hide or show views using setVisibility(int).

CREATING A BUTTON

Views may have an integer id associated with them.

These ids are typically assigned in the layout XML files, and are used to find specific views within the view tree.

- Define a Button in the layout file and assign it a unique ID.

```
<Button
    android:id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

- From the onCreate method of an Activity, find the Button

```
Button myButton = (Button) findViewById(R.id.my_button);
```

https://developer.android.com/guide/topics/ui/controls/button.html

CREATING A BUTTON

Views may have an integer id associated with them.

These ids are typically assigned in the layout XML files, specific views within the view tree.

- Define a Button in the layout file and assign it a unique ID.

```
<Button
    android:id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

- From the onCreate method of an Activity, find the Button

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object (for example, TextView supports the textSize attribute), but these attributes are also inherited by any View objects that may extend this class.

https://developer.android.com/guide/topics/ui/controls/button.html

CREATING A BUTTON

Views may have an integer id associated with them.

These ids are typically assigned in the layout XML files, and are used to find specific views within the view tree.

- Define a Button in the layout file and assign it a unique ID.

```
<Button
    android:id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

- From the onCreate method of an Activity,

```
Button myButton = (Button) f
```

The at-symbol (@) at the beginning of the id string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource. The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the R.java file).

https://developer.android.com/guide/topics/ui/controls/button.html

RESPONDING TO EVENTS

When the user clicks a button, the Button object receives an on-click event.

Add the android:onClick attribute to the <Button> element in your XML layout.

The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

RESPONDING TO EVENTS

You can also declare the click event handler programmatically rather than in an XML layout.

This might be necessary if you instantiate the Button at runtime or you need to declare the click behavior in a Fragment subclass.

To declare the event handler programmatically, create an View.OnClickListener object and assign it to the button by calling setOnClickListener(View.OnClickListener).

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

COMMON EVENT LISTENERS

onClick(): From View.OnClickListener. This is called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys and presses the suitable "enter" key or presses down on the trackball.

onLongClick(): From View.OnLongClickListener. This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).

onFocusChange(): From View.OnFocusChangeListener. This is called when the user navigates onto or away from the item, using the navigation-keys.

onKey(): From View.OnKeyListener. This is called when the user is focused on the item and presses or releases a hardware key on the device.

onTouch(): From View.OnTouchListener. This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).

VIEW ATTRIBUTES

E.g. a RadioButton

android:checked="***bool***"	set to true to make it initially checked
android:clickable="***bool***"	set to false to disable the button
android:id="@+id/***theID***"	unique ID for use in Java code
android:onClick="***function***"	function to call in activity when clicked (must be public, void, and take a View arg)
android:text="***text***"	text to put next to the button

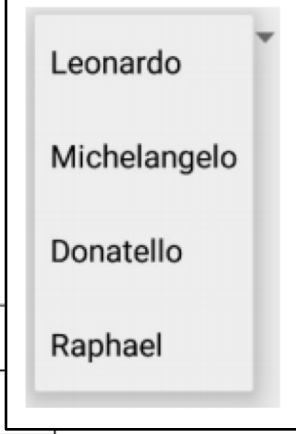
```
<RadioGroup ...
        android:orientation="horizontal">
    <RadioButton ... android:id="@+id/lions"
                     android:text="Lions"
                     android:onClick="radioClick" />
    <RadioButton ... android:id="@+id/tigers"
                     android:text="Tigers"
                     android:checked="true"
                     android:onClick="radioClick" />
    <RadioButton ... android:id="@+id/bears"
                     android:text="Bears, oh my!"
                     android:onClick="radioClick" />
</RadioGroup>
```

```
// in MainActivity.java
public class MainActivity extends Activity {

    public void radioClick(View view) {
        // check which radio button was clicked
        if (view.getId() == R.id.lions) {
            // ...
        } else if (view.getId() == R.id.tigers) {
            // ...
        } else {
            // bears ...
        }
    }
```

https://developer.android.com/guide/topics/ui/controls/radiobutton.html

VIEW ATTRIBUTES

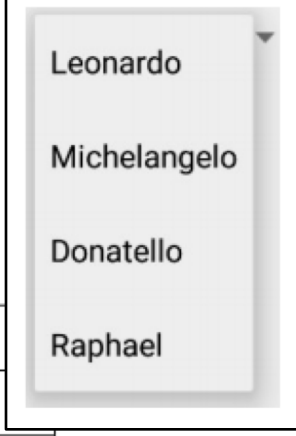


E.g. a Spinner

android:clickable="***bool***"	set to false to disable the spinner
android:id="@+id/***theID***"	unique ID for use in Java code
android:entries="@array/***array***"	set of options to appear in spinner (must match an array in strings.xml)
android:prompt="@string/***text***"	title text when dialog of choices pops up

```
<LinearLayout ...>
    <Spinner ... android:id="@+id/tmnt"
        android:entries="@array/turtles"
        android:prompt="@string/choose_turtle" />
    <TextView ... android:id="@+id/result" />
</LinearLayout>
```

VIEW ATTRIBUTES



E.g. a Spinner

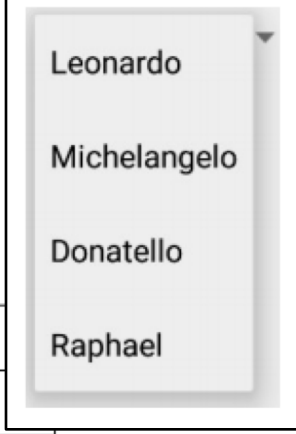
android:clickable="***bool***"	set to false to disable the spinner
android:id="@+id/***theID***"	unique ID for use in Java code
android:entries="@array/***array***"	set of options to appear in spinner (must match an array in strings.xml)
android:prompt="@string/***text***"	title text when dialog of choices pops up

```
<LinearLayout ...>
    <Spinner ... android:id="@+id/tmnt"
        android:entries="@array/turtles"
        android:prompt="@string/choose_t
    <TextView ... android:id="@+id/resul
</LinearLayout>
```

The name of an XML element for a view is respective to the Android class it represents.

So a <TextView> element creates a TextView widget in your UI, and a <LinearLayout> element creates a LinearLayout view group.

VIEW ATTRIBUTES



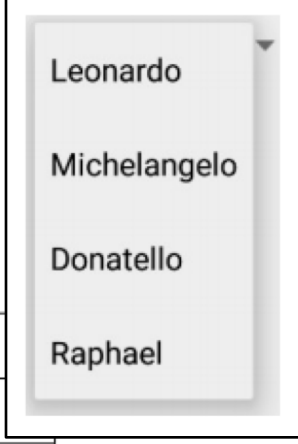
E.g. a Spinner

android:clickable="***bool***"	set to false to disable the spinner
android:id="@+id/***theID***"	unique ID for use in Java code
android:entries="@array/***array***"	set of options to appear in spinner (must match an array in strings.xml)
android:prompt="@string/***text***"	title text when dialog of choices pops up

in res/values/strings.xml:

```
<resources>
    <string name="choose_turtle">Choose a turtle:</string>
    <string-array name="turtles">
        <item>Leonardo</item>
        <item>Michelangelo</item>
        <item>Donatello</item>
        <item>Raphael</item>
    </string-array>
</resources>
```

VIEW ATTRIBUTES



E.g. a Spinner

`android:clickable="`***`bool`***`"`	set to false to disable the spinner
`android:id="@+id/`***`theID`***`"`	unique ID for use in Java code
`android:entries="@array/`***`array`***`"`	set of options to appear in spinner (must match an array in `strings.xml`)
`android:prompt="@string/`***`text`***`"`	title text when dialog of choices pops up

```
// in MainActivity.java
public class MainActivity extends Activity {
  ...
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Spinner spin = (Spinner) findViewById(R.id.tmnt);
    spin.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
      public void onItemSelected(AdapterView<?> spin, View v, int i, long id) {
        TextView result = (TextView) findViewById(R.id.turtle_result);
        result.setText("You chose " + spin.getSelectedItem());
      }

      public void onNothingSelected(AdapterView<?> parent) {}  // empty
    });
  }
}
```

XML VS RUNTIME

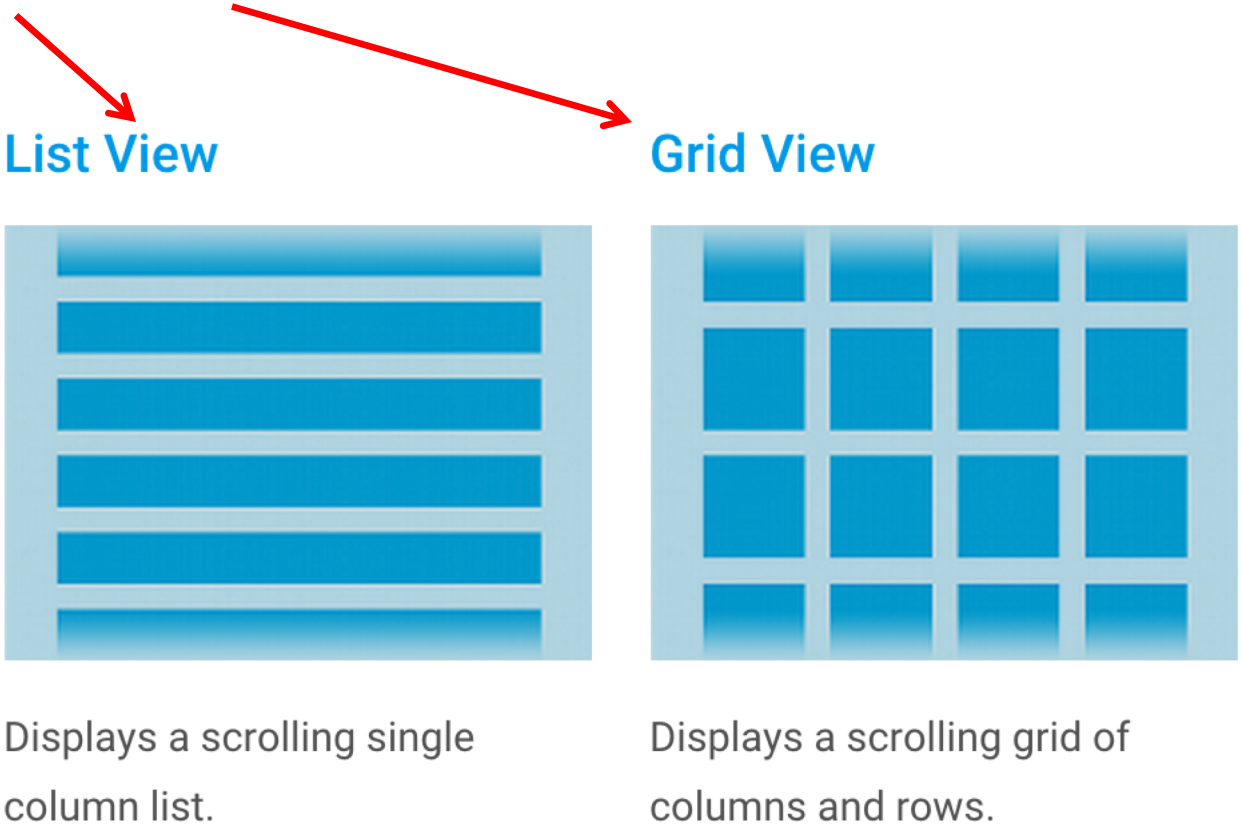
Advantages to declaring your UI in XML include:

- It enables you to better separate the presentation of your application from the code that controls its behavior.
- Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile.
- E.g. you can create XML layouts for different screen orientations, different device screen sizes, and different languages.
- Declaring the layout in XML can make it easier to visualize the structure of your UI, so it's easier to debug problems.
- XML is compiled into very efficient form for runtime, so performance is not a problem

DYNAMIC LAYOUTS

When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses AdapterView to populate the layout with views at runtime.

A **subclass of the AdapterView** class uses an Adapter to bind data to its layout.



LAYOUT ADAPTERS

You can populate an AdapterView such as ListView or GridView by binding it to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

- The Adapter behaves as a middleman between the data source and the AdapterView layout.
- It retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the AdapterView layout.
- By acting as a middleman Adapters enable layouts such as ListView to be used with many different types of data and views, e.g. images vs text.

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView.

ARRAYADAPTER

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.

E.g. if you have an array of strings you want to display in a ListView initialize a new ArrayAdapter using a constructor to specify the layout for each string and the string array:

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

The arguments for this constructor are:

- Your app Context
- The layout that contains a TextView for each string in the array
- The string array

ARRAYADAPTER

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.

E.g. if you have an array of strings you want to display in a ListView initialize a new ArrayAdapter using a constructor to specify the layout for each string and the string array:

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

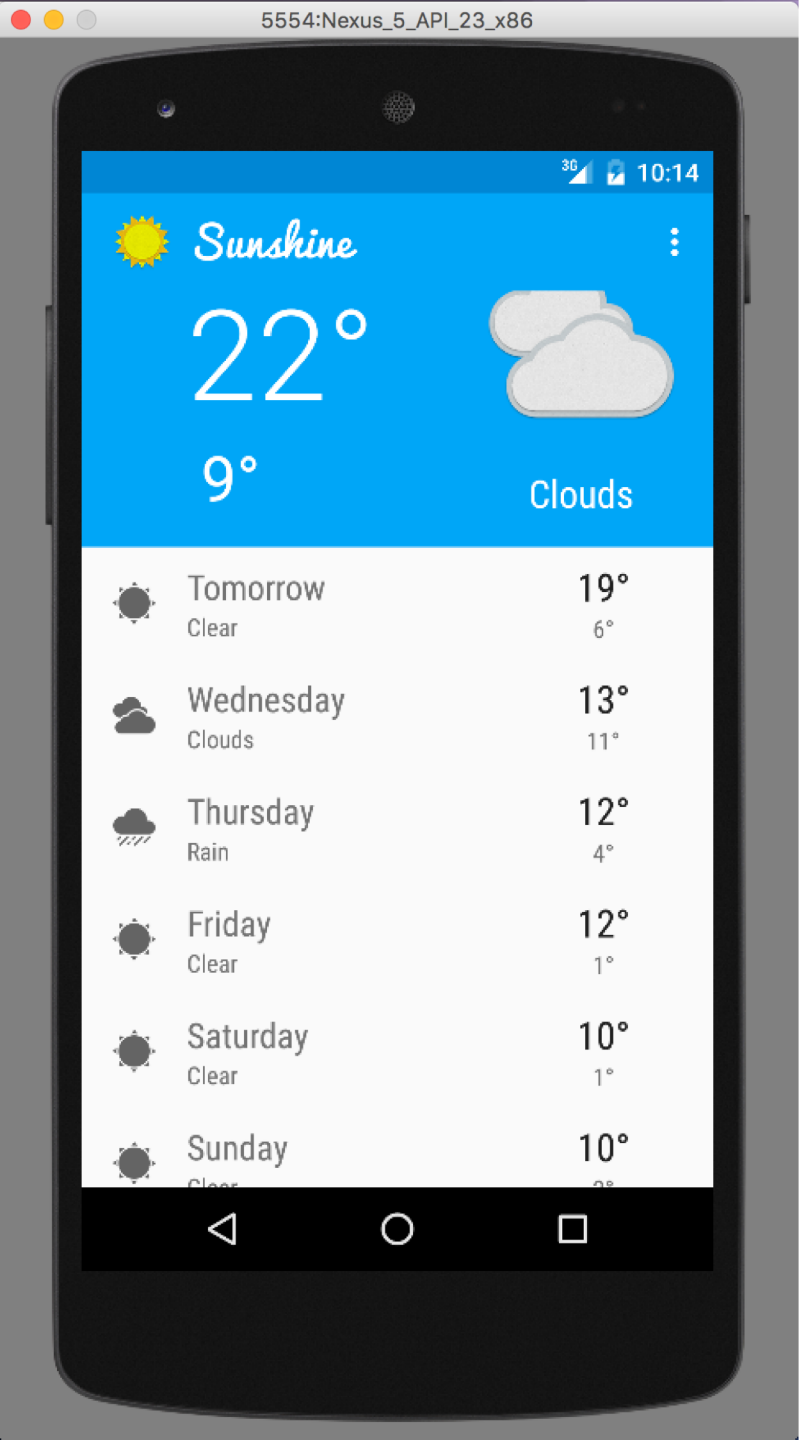
Then call setAdapter() on your ListView:

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

FURTHER DETAILS

Android user interface guides:

https://developer.android.com/guide/topics/ui/index.html



Touches on the following:

- View and ViewGroup classes
- Layouts
- Threading with AsyncTask
- Networking, e.g. using a web service

QUESTIONS?

Contact:

d.coyle@ucd.ie

Please ask in the Discussion Forum.

Tutorial:

Weather app part 1