

Android

Lesson 2 by Callum Taylor

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

Introduction

- All code and presentation slides can be found over at <https://github.com/scruffyfox/AndroidCourse>
- Twitter/app.net/github: @scruffyfox
- [http://\(blog.\)callumtaylor.net](http://(blog.)callumtaylor.net)

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>


  @scruffyfox

Introduction

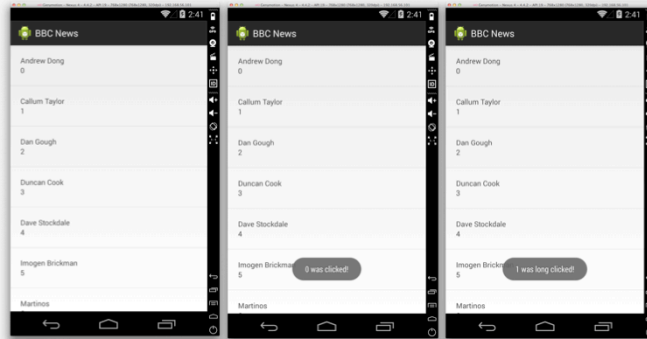


<https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

What we're going to make



Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>


  @scruffyfox

This is what we're going to make today

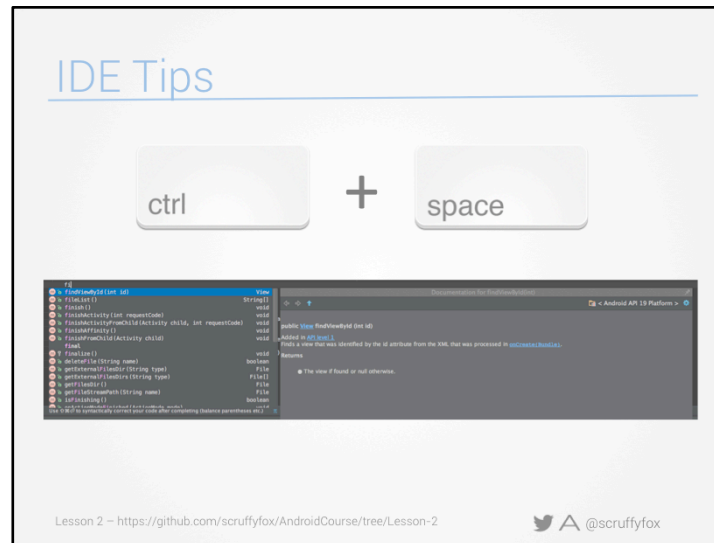
What we're going to make

- ListView
- ArrayAdapter/BaseAdapter
- findViewById
- View Holder paradigm

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

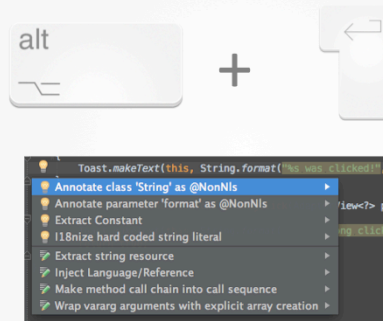
  @scruffyfox

This is what we're going to make today



Control + space will be the most popular shortcut you use, pressing it will bring up the codesense suggestion dialog when you type to allow you to find the method/variable you're looking for. It will also bring up the documentation for the highlighted method. Press enter to auto complete

IDE Tips



Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

Pressing `alt` and `enter` will open up the quick fix menu which will give you some useful functions such as refracting a string or reversing an if statement



Pressing control and enter will bring up the generate menu which is useful when overriding methods from the super class you don't know the name of

IDE Tips



```
55 @NotNull Context context, CharSequence text, @MagicConstant int duration  
56 @NotNull Context context, @ResourceInt int resid, @MagicConstant int duration  
57 Toast.makeText(this, |  
58 return true;
```

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

Pressing control and p will bring up the parameters for a method as you're typing.


How R.java works

```
package net.calluntaylor.news;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int list_view=0x7f060002;
        public static final int name=0x7f060000;
        public static final int position=0x7f060001;
    }
    public static final class layout {
        public static final int list_item=0x7f030000;
        public static final int main_view=0x7f030001;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
    public static final class style {
        // Customize your theme here.
        public static final int AppTheme=0x7f050000;
    }
}
```

R.layout.list_item
R.id.name
R.drawable.ic_launcher

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

Because Android is made up of 2 parts, java source and XML resources, there needs to be a way for the java code to access and use the xml resources. The way that this is handled is by using a file called R.java.

R.Java is automatically generated for you and lives under the build/source/r folder.

Every XML resource is given an ID which you can reference in code using the static ints found in R like so

How R.java works

```
((TextView)findViewById(R.id.name)).setText(  
    // setText(char[] text, int start, int len) void  
    // setText(CharSequence text) void  
    // setText(int resid) void  
    // setText(int resid, BufferType type) void  
    Toast.makeText(this, String.format("Hello World!"), Toast.LENGTH_SHORT).show();  
    // setText(int resid, BufferType type) void
```

```
MainActivity.java x strings.xml x AndroidManifest.xml x  
1 <resources>  
2   <string name="app_name">BBC News</string>  
3   <string name="hello_world">Hello World!</string>  
4 </resources>  
5
```

R.string.hello_world

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

 @scruffyfox

When writing code, 9 times out of 10, the method which you're using to change the UI will have at least 2 overflows, one for straight input via code, and one for reference to a resource

Here we have the option to either write in text manually, or reference a resource file which would be stored in the strings.xml file such as the "hello_world" field we used before

How Adapters work

- getCount()
- getItem(int position)
- getItemId(int position)
- getView(int position, View convertView, ViewGroup parent)

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

Adapters are very clever in its design of serving a large data set to the UI of an app. The pattern consists of a few core methods.

How Adapters work

- `getCount()`

Exactly what it says – returns the number of items in the current data set

How Adapters work

- `getItem(int position)`

Returns the item from the data set at the specific position

How Adapters work

- `getItemId(int position)`

Returns the item's id at the position. Can just return 0, but return a reliable int ID/representation of the item to increase performance of the list view

How Adapters work

- `getView(int position, View convertView, ViewGroup parent)`

Returns the view in the list. This is where we do all of our data inflating into the screen which the user sees.

Position is the position of the item in the dataset

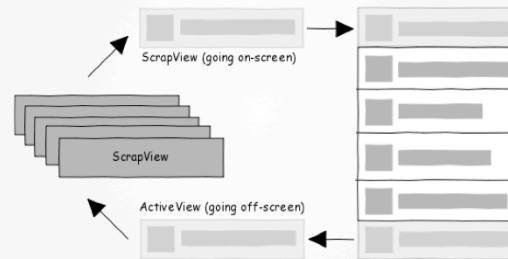
convertView is the view that will be shown

Parent is the parent view of the view being shown (usually the list view)



Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

How List Views work



Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

List views are very clever with how they show the views on the screen. If you have a data set of say 10,000 items, obviously to have 10,000 views on the screen would cause the phone to firstly crash, but also to be **EXTREMELY** slow. The way a list view works to solve this is called **view recycling**.



When you scroll up or down, the view at the top or bottom drops off the screen, when it is no longer visible, it will get put into a scrap view heap, and when a new view is shown on the screen, that view is recycled from the stack, back on to the screen. This is why when we only set the views when convertView is null, it will jump around as you scroll.

How List Views work

- `getView(int position, View convertView, ViewGroup parent)`

```
@Override public View getView(int position, View convertView, ViewGroup parent)
{
    if (convertView == null)
    {
        convertView = LayoutInflater.from(context).inflate(R.layout.list_item, parent, false);
    }
    ((TextView)convertView.findViewById(R.id.name)).setText(getItem(position));
    return convertView;
}
```

Lesson 2 – <https://github.com/scruffyfox/AndroidCourse/tree/Lesson-2>

  @scruffyfox

In our adapter, this is where `convertView` comes in handy.

When a new view is being created, `convertView` will be null. Here it will be safe to create our view and return it. But when `convertView` is **not** null, that is the view being recycled, so we don't have to create another view, we can just reuse the same view but change the contents to reflect the data.