

# KCL Tech Build X: Android

## Lecture Two: Layouts, Views and Events

- Recap of Last Week (quick)
- Layout Resources
- Parts of a Layout: Views, View Groups & Other Resources
- Listening to View Actions

### Recap of Last Week

#### Environment Setup

You should have **Android Studio** installed and set up, ready to run on your machine. You should also have the **sample app** downloaded and be able to open it in Android Studio. If you don't already have it, be aware that **we will not be downloading/installing Android Studio in this lecture** - that should have been done already in the first lecture, or in the follow-up drop in session.

#### KCL Tech Todo

Brief reminder: we will build a **simple todo list app** with a few basic features: add tasks, edit tasks and complete tasks. The tasks have a title, due date and optional notes field. This is the “bare minimum” we will produce, but if we speed through the course quickly then we will build more!

A completed version of this “minimum app” can be **downloaded from Google Play** on an Android device by visiting [<http://tiny.cc/kcl-tech-todo>] or scanning the QR code.



#### Basic App Components

- **Activities**: a “page” in your app; these have the “create, resume, pause, stop” lifecycle.
- **Resources**: files that store the *stuff* your app uses: strings, numbers, colours, layouts, etc.
- `AndroidManifest.xml`: a spec-sheet for your app, detailing the things it needs, the things it contains and the things it can do.
- **Layouts**: a type of resource that store the **structure and appearance** of your app's components.

### Layout Resources

As we know, resources are **XML files** that contain information your app will use. This can be strings the user will see, colours and dimensions for layouts, the actual layouts, etc. We'll focus on **layouts** in this lecture.

The idea and purpose behind layouts is straightforward: they just tell your device “*put this here, put that there, put this under that, this should be as tall as that, etc.*” until the screen looks how you want it to.

In order to tell Android *which* layout to use for *which* part of your app, you need to know how to refer to them. Layouts all live in a specific folder in your app project: `app/res/layout`.

Right now we only have one activity and one layout. How does the activity know which layout to use? Open the file, and you'll see that it just takes this one line of code:

```
setContentView(R.layout.activity_hello_world);
```

Broken down, here's the meaning of each part of that line:

```
setContentView(
```

We're setting the view for this activity.

```
R.
```

We want a resource...

```
layout.
```

...it's a layout...

```
activity_hello_world
```

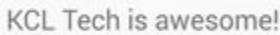

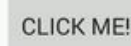
...and this is its name.

```
);
```

Done!

## Parts of a Layout: Views

As we said last week, layouts are made up of many **views**. A view is an individual "thing" that a user will interact with. We'll be adding views to our layouts in **XML**. We're going to start off with four common views:

<code>&lt;TextView /&gt;</code>	
<code>&lt;EditText /&gt;</code>	
<code>&lt;Button /&gt;</code>	
<code>&lt;ImageView /&gt;</code>	

Each view has a number of **attributes** to define everything about it: how big it is, what colour it is, what text it displays, etc. These are explicitly stated in the XML you'll write (think back to "XML attributes" from last week). Every view has **at least these two attributes**:

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

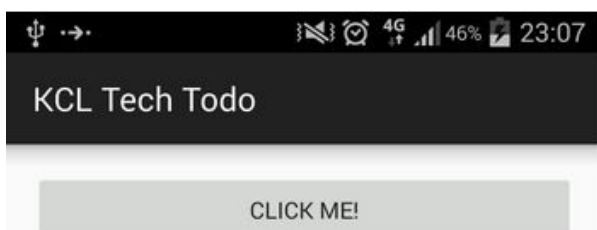
It should be quite obvious what they mean: how wide is this view, and how high is it? You could put a number of pixels in here, or a measurement in one of the other dimension units that Android supports (more on them later), but instead we're going to use two **special keywords**.

The two keywords are `match_parent` and `wrap_content`. Both of these can be used for width or height, and you can use them in any combination. Their meanings are as follows:

- `match_parent`: make this view as wide/tall as the view group surrounding it.
- `wrap_content`: make this view just wide/tall enough to show what it contains, but no more.

The button width is `match_parent`:

The button width is `wrap_content`:



## Exercise

Add one or more of the first three views to the “Hello World” layout you already have. You’ll need to use the tag names above, the two attributes mentioned, as well as the `android:text` attribute for `TextViews` and `Buttons`, and the `android:hint` attribute if you want something in the background of your `EditText`.

Leave the `ImageView` for now, but make sure you try it later. Post any questions you have and the results you get on the `#android-programming` channel of [<http://kcltechhq.slack.com>].

## Parts of a Layout: View Groups

Some special views can contain other views inside them; these are called **view groups**. The views *inside* a view group are called **children**; the view group *surrounding* a view is called its **parent**. Generally speaking, **you never see a view group**: they don’t display anything of themselves, they are just used to organise their children (which you *can* see).

There are many types of view group; we’re going to look at the **three most common ones**.

**Side note:** two of the view groups we’ll look at have names that end with “layout”, just to confuse us. For clarification, we’re stick to this definition throughout the course: a layout is an XML resource file containing many views and view groups; a view group is a type of view that can hold other views.

### ScrollView

This is absolutely **the simplest** view group that exists. It only has one child (often another view group), and it does one thing with it: **adds a scrollbar** if it gets too tall/wide. *That’s it*. You can put a view group inside a `ScrollView`, and if it’s small enough to fit on the screen then you won’t even know the `ScrollView` is there; if it’s too big, you’ll get a little scrollbar. Simple as that.

### LinearLayout

Again, this view group is pretty simple, because it can do only one thing: **stack views**. It can stack them **horizontally or vertically**, but that is all it can do. You can specify horizontal or vertical with the `android:orientation` attribute.

If you need to, you can control the `android:gravity` attribute inside a `LinearLayout` to make everything “float” to the middle, left, right, etc., but that’s not required - if you don’t specify, the view group’s layout will default to “pull to top-left”.

### RelativeLayout

This view group takes a big step up in complexity, but you get **a lot more power** to play with. Instead of simply putting views in a single-file column or row, this lets you specify structures like “put this view *to the left of* that view; put this view *underneath* that view; make this view line up with the right edge of the view over there; etc.”.

We won’t be using a `RelativeLayout` in this lecture, but we *will* be using them in the project. You’ll find a short exercise on `RelativeLayouts` on the resource site [<http://android.kcl.tech>] - give it a try; it should take no longer than 15 or 20 minutes.

## Parts of a Layout: Other Resources

Last week we covered **other resources**, like strings (chunks of text), dimensions, styles, etc. You can find details on these in last week's notes, or in the code from the blank project app you should have access to.

Refer to these files to follow along with the slides:

- Strings: `app/res/strings.xml`
- Dimensions: `app/res/dimens.xml`
- Styles: `app/res/styles.xml`

## Exercise

Using the views we've discussed and a vertical `LinearLayout`, try to construct the "Edit Task" activity by adding and changing the "Hello World" activity you already have.

## Listening to View Actions

Views are pretty, but they're also **useless** if we can't do anything with them. They exist for users to interact with, so let's listen for a user's action. We'll **start simple** by doing something when a user clicks a button.

Before we can tell Java to do something with the button, we need to **get a reference to it**. In Android, this is easy! First, we **give the view an ID** by adding this attribute: `android:id="@+id/my_button"`. Then we can tie the button to a variable in Java with this code:

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

Now that we have `myButton`, we can do things with it. We want to listen out for click events, so we're going to add an `OnClickListener`, like this:

```
myButton.setOnClickListener(new OnClickListener() {  
    public void onClick(View v) {  
        // do something!  
    }  
});
```

Now, whenever someone clicks on your button, it's going to run the code inside your `onClick` method. Try each of these lines one by one and see what happens:

```
Toast.makeText(getApplicationContext(), "Hello!", Toast.LENGTH_SHORT).show();
```

```
finish();
```

```
v.setVisibility(View.GONE);
```

## What Next?

- Fill in our survey under "Lecture Two" on [<http://android.kcl.tech>]
- Try the `RelativeLayout` exercise on [<http://android.kcl.tech>]
- Try to use an `ImageView`
- Sign up to [<https://kcltechhq.slack.com>]