KCL Tech Build X: Android

Lecture Three: Events, Intents and the Activity Lifecycle

- Recap of Last Week (quick)
- Listening to View Actions
- Using Intents
- The Activity Lifecycle

Recap of Last Week

Progress So Far

- You should have Android Studio installed and set up, ready to run on your machine.
- You should have the blank project downloaded and opened.
- You should have adapted the "Hello World" activity to look like the "Edit Task" activity.

Layouts, Views and View Groups

- A **layout** is a specific resource that tells Android how part of your app is structured and how it looks.
- A **view** is an individual component of a layout, like a button or an image.
- A **view group** is a special type of view that can hold and organise other views inside it.

Listening to View Actions

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

First we need to get a reference to the button. 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 like this:

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

Now that we have `myButton` as a variable, 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!
    }
});
```

When the button is clicked, the code inside `onClick` will run. Here are a few options you could try out:

```
Toast.makeText(getApplicationContext(), "Hello!", Toast.LENGTH_SHORT).show();
```
This will display a small pop-up, or "toast", with the message "Hello!".

```
finish();
```
This will close or "finish" the activity you are using.

```
v.setVisibility(View.GONE);
```
The variable v represents our button, so this will hide it completely.

Intents

An intent represents an app's **intent to do something**. Starting an intent tells the operating system (Android) "hey, I want to do something now". You app can "intend to do" any number of actions: go to a new activity, share a photo, send a message, etc. You can even define your own custom actions!

We'll jump right into it and **start simple**: we'll make an intent that starts another activity.

```
Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);
```

Important: at the top of your Java file don't forget to add `import android.content.Intent;` otherwise you'll get errors about not being able to resolve a symbol! In Android Studio, press `Alt + Enter` (or `Option + enter` on Mac) to import missing classes automatically.

Just creating an intent won't actually do anything. You need to **start** it first! Doing so is really easy - in this case you just call the method `startActivity()` and pass your intent as a parameter. See below:

```
Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);
startActivity(intent);
```

Sending Extras

What if you wanted to **send some information** with that intent? Easy - you can send **extras** with any intent.

Any extra is a simple **key/value pair** - you can specify a **value** and a **name (key)** to associate with it. The key will always be a **string**, the value can be any **primitive type** (that means a string, a number or a boolean). The value can actually be a couple of other things, but we won't be covering them yet.

Adding an extra to an intent is easy:

```
Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);
intent.putExtra("my_key_1", "value 1");  // adding a string extra
intent.putExtra("my_key_2", 42);         // adding a number extra
intent.putExtra("my_key_3", true);       // adding a boolean extra
startActivity(intent);
```

There are a few **predefined keys** that you can use with internal intents, such as `EXTRA_MESSAGE`.

Reading Extras

Once you've sent an intent with some extra data, how you need to **read them back** to be able to use them.

No matter how they're invoked (by the user or by the system), **every activity** is launched with an intent. You can get the intent that an activity was started with from the `getIntent()` method. A good place to call this is in the activity's `onCreate()` method (more on that in the next section).

Once you have the intent, you can use the method `getExtras()` to get all of the extras that were sent with the intent. This method returns a `Bundle` object, which contains all of the key/value pairs that were sent. The `Bundle` class is widely used in Android, so it's worth reading up on the documentation for it (see link on [http://android.kcl.tech]).

> **Caution!** The return from `getExtras()` could be null, so you need to check for that!

To retrieve the values from your bundle, you can use the following methods:

```java
Intent intent = getIntent();
Bundle extras = intent.getExtras();
if (extras != null) {
    String myValue1 = extras.getString("my_key_1");
    int myValue2 = extras.getInt("my_key_2");
    boolean myValue3 = extras.getBoolean("my_key_3");
}
```

The Activity Lifecycle

Reminder: an activity is analogous to a "page" within your app. You create an activity by creating a Java class that extends `Activity`, `AppCompatActivity`, or some other class from the `Activity` family.

Why Does an Activity Need a Lifecycle?

Activities are, usually, the biggest and most widely-used building blocks in an application. You users will spend the vast majority of their time in your app interacting with an activity, and they tend to be responsible for a lot of complex operations.

Because of all that, it's important that your activities know exactly what's happening to them. It's no good if your activity keeps playing a video after the user closes it, or if it doesn't know how to start a game when the user opens it!

What *is* the Lifecycle?

During its lifetime, an activity will move between several states. Each time an activity switches from one state to another, it will call a method to tell you about the state it's entering. For example, when the activity enters the state "paused", it will call the method `onPause()`.

> **What does *"state"* mean?** The easiest way to grasp the concept of a *state* is with examples. Consider a light bulb; it has two states: "on" and "off". Your laptop probably has around three states: "on", "standby" and "off". Activity states are the same, except there are more of them.

We're interested in the following states of an activity:

- Created
- Started
- Resumed
- Paused
- Stopped
- Destroyed

The movement between states is depicted in the diagram on the next page.

Source: [https://android.developer.com]
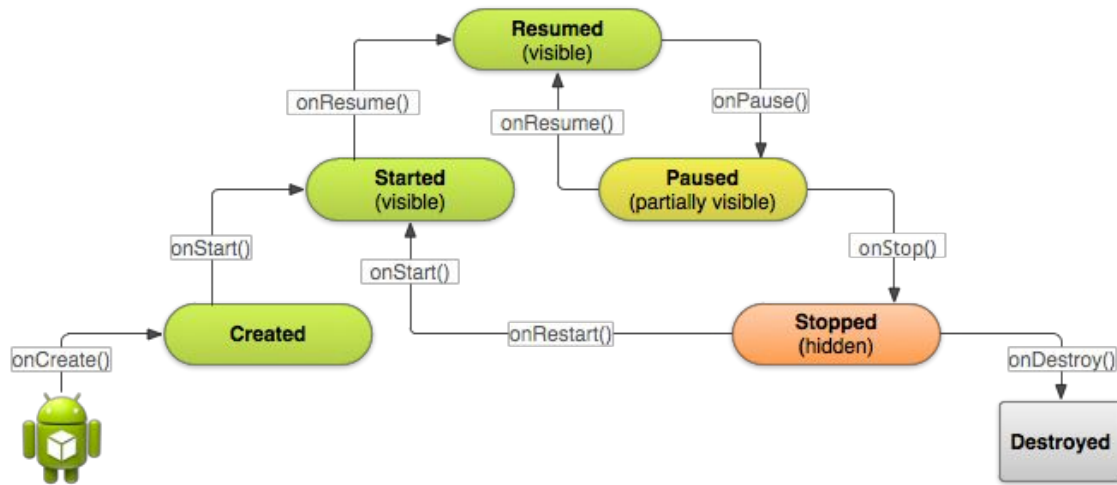
There are several flows through this system, depending on what the user is doing. Each stage will be explained on the slides, but here are some example flows:

- When opening the app:
  `onCreate()` → `onStart()` → `onResume()`

- When the back button is pressed and the app is exited:
  `onPause()` → `onStop()` → `onDestroy()`

- When the home button is pressed:
  `onPause()` → `onStop()`

- After the home button is pressed and the app is opened from recents list or the icon:
  `onRestart()` → `onStart()` → `onResume()`

- When another app is opened from the notification bar:
  `onPause()` → `onStop()`

- When the back button is pressed from another app and our app comes back into view:
  `onRestart()` → `onStart()` → `onResume()`

- When any dialog opens on the screen:
  `onPause()`

- After dismissing the dialog:
  `onResume()`

- When the phone screen is turned off:
  `onPaused()` → `onStop()`

- When the screen comes on again:
  `onRestart()` → `onStart()` → `onResume()`

What Next?

- Next week is **reading week** - don't turn up, because we won't be here! (We will be on **Slack** though.)
- Finish off the lifecycle activity in the slides (use **Slack** for help!).
- Maria and Mark will post some exercises and content in the **Slack** channel.
- After the break: `ListView`s, databases and putting it all together.