KCL Tech Build X: Android

Lecture Four: Recap & ListViews

- Recap of Progress So Far
- ListViews
- Homework

Recap of Progress So Far

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.

Events

- Once you acquire a reference to a view with `findViewById(R.id.xyz)`, you can use methods such as `setOnClickListener(...)`, `setOnLongClickListener(...)`, etc. to "listen" to the various ways a user can interact with it.

Intents

- An intent tells the operating system that your apps *intends* to do something.
- Most commonly these are used to move to another activity, but they can also be used to start a phone call, share a file… almost anything!
- For code samples, check last week's notes.

Activity Lifecycle

- Activities go through a complex lifecycle, which are used to keep your app updated on its status. Is it open? Can the user see it? Has it been closed? All of those events and more are part of the lifecycle.

ListViews

ListViews are, undoubtedly, the most widely used **complex view** (views like a Button or TextView are considered to be more basic views). As is often the case in Android, they do exactly what you think they do: they **display a list of things**.

Why Are They Needed?

In terms of memory, creating and displaying views is **reasonably expensive**. A view usually consumes a couple of kilobytes of RAM, and when you have complicated layouts this cost can really add up! Imagine you have 100 contacts in the phone book, and each one has a name, number, picture, and probably a few other views and view groups - that's already a few megabytes of memory your app has taken up, before any computation work has actually happened.

ListViews help you to avoid this problem by **recycling** views: as the user scrolls down, views that move off the top of the screen are recycled and inserted into the list further down; the process is reversed as the user scrolls down.

The example app (see [http://android.kcl.tech]) and the slide demonstration should make this concept clear.

Adapters

A ListView is mapped to a **data source** - a list of contacts, the tasks in our todo app, etc. The bridge between the data and your view is called the **adapter**. An adapter is just a Java class that has certain methods, used by a ListView.

Adapters are used to help with **separation of concerns** in your software, and there are two important effects of this:

Your data **does not know, and does not care**, that it is being displayed in a ListView.

The ListView **does not know, and does not care**, what your data represents.

The process is simple: you give your data to the adapter, and the ListView uses the adapter to retrieve your data (and information about it).

But why? Following this pattern, you can build a **custom adapter** to accept whatever type of data you have, and that adapter can expose a **pre-defined interface** that a ListView can understand.

To create an adapter you need to extend one of the base adapter classes - we're going to be using `BaseAdapter`, but there are others - and then start overriding methods that give information about your data. The three methods we're going to override are:

- `getCount()` - how many items should be in the list?
- `getItem(int p)` - give me the **item** that you want to display at position `p`.
- `getView(int p)` - give me the **view** for the item you want to display at position `p`.

These methods will be explained in more details during a live-code session in the lecture, and a fully detailed explanation can be found at [http://tiny.cc/base-adapter]. Once you've created your adapter class, you need to assign it to your ListView. This is easy:

```
// get a reference to your ListView (you may have a different ID)
ListView listView = (ListView) findViewById(R.id.list_view);

// create an adapter (using whatever arguments you specified)
SpecialAdapter adapter = new SpecialAdapter( ... );

// link them together
listView.setAdapter(adapter);
```

Changing Data at Runtime

You've made your ListView, connected it to your adapter, and everything is great! What if you want to **change the data** inside the list, or add/remove items? The adapter class has you covered!

You can **create your own methods** within your adapter class that change the data in any way you like. They could replace the data source, empty it, etc. The only thing you need to remember to do is notify your adapter when you change the data, by simply calling `notifyDatasetChanged()`.