## Android Studio Fundamentals Notes 01.2: Basic app anatomy

## Activities

* MainActivity is a subclass of AppCompatActivity, which in turn is a subclass of Activity. An Activity is a core Android class that is responsible for drawing an Android app UI and receiving input events.
* All activities have an associated layout file, which is an XML file in the app's resources. The layout file is named for the activity, for example, activity\_main.xml.
* The setContentView() method in MainActivity associates the layout with the activity and inflates that layout when the activity is created.
* Layout inflation is a process where the views defined in the XML layout files are turned into (or "inflated" into) Kotlin view objects in memory. Once layout inflation happens, the Activity can draw these objects to the screen and dynamically modify them.

## Views

* All UI elements in the app layout are subclasses of the View class and are called *views*. TextView and Button are examples of views.
* View elements can be grouped inside a ViewGroup. A view group acts as a container for the views, or other view groups, within it. LinearLayout is an example of a view group that arranges its views linearly.

## View attributes

* The android:layout\_width and android:layout\_height attributes indicate the weight and height of a view. The match\_parent value stretches the view to its parent's width or height. The wrap\_content value shrinks the view to fit the view's contents.
* The android: text attribute indicates the text that a view should display (if that view displays text.) For buttons, android:text is the button label.
* The android:orientation attribute in a LinearLayout view group arranges the view elements it contains. A value of horizontal arranges views left to right. A value of vertical arranges the views top to bottom.
* The android:layout\_gravity attribute determines the placement of a view and all that view's children.
* The android:textSize attribute defines the size of the text in a text view. Text sizes are specified in sp units (*scalable pixels*). By using sp units, you can size text independently of the device's display quality.

## Strings

* Instead of hardcoding strings in the layout, it's a best practice to use string resources.
* String resources are contained in the values/res/string.xml file.
* To extract strings, use Alt+Enter (Option+Enter on a Mac). Select Extract string resources from the popup menu.

## Using views

* To connect your Kotlin code to a view that you defined in the layout, you need to get a reference to the view object after the view has been inflated. Assign an ID (android:id) to the view in the layout, then use the findViewById() method to get the associated view object.
* When you create an ID for a view in the XML layout file, Android Studio creates an integer constant with that ID's name in the generated R class. You can then use that R.id reference in the findViewById() method.
* You can set the attributes of a view object in your Kotlin code directly by property name. For example, the text in a text view is defined by the android:text attribute in the XML, and it is defined by the text property in Kotlin.
* A *click handler* is a method that is invoked when the user clicks or taps on a UI element. To attach a click-handler method to a view such as a button, use the setOnClickListener() method.

## Using toasts

A toast is a view that shows the user a simple message in a small popup window.

To create a toast, call the makeText() factory method on the Toast class with three arguments:

* The context of the app Activity
* The message to display, for example a string resource
* A duration, for example Toast.LENGTH\_SHORT

To display the toast, call show().