Title: Activities and Basic UI building

1. Activity
   1. In simple words an Activity represents a user’s screen. It’s a single focused thing a user can do. The activity class takes care of creating the screen and setting it up for us. In some rare cases one screen might have more than one activity.

E.g. the example project that we built had a MainActivity class, which is the first activity that gets opened when the user starts our application. Open this file and you will see a line like this inside the onCreate function:

setContentView(R.layout.*activity\_main*);

This line loads the layout file activity\_main.xml and sets it as this Activity’s view. We will discuss onCreate in a lot of detail, later in these notes.

1. Building basic UI: Open the activity\_main.xml file, which is present inside the layout subfolder of res directory. You will see two tabs at the bottom of file editor. Lets discuss these tabs one by one.
   1. Graphical Layout: Click on the Graphical layout and you will see a bunch of sub-screens, lets discuss the important once:
      1. Preview: The middle portion of the screen shows how the screen will look like with this layout. Right now it should be showing you “Hello World!” at top left of the screen.
      2. Palette: As what the name suggests, you will find various different elements that can be added to the layout. You can find Radio Buttons, Text Views, Images etc in this subview. You can drag and drop any of these in the Preview subscreen and add them to our layout file.
      3. Structure: This consists of two parts. Top part just shows you various components that are currently present in the layout file. When you select one of the added components you will see that component’s properties in the lower half. You can change a bunch of properties like text size, color, height, width etc from this window.

Exercise: Add couple of textfields and change their text to 1 and 2. Change the font size and color from the properties window for both of them.

* 1. XML: This tab shows us the actual content of the file. Graphical layout is just a fancier way of editing this file. After gaining some experience, editing XML file directly will be a lot quicker compared to doing changes in the Graphical layout.

Lets look at some important properties that you will use again and again:

* + 1. Id: This is the handle we will use to find a specific view during runtime. We will be able to make changes in a view by using this id.
    2. Height and Width: One of the most important category of properties. You can define the height and width of your views by changing layout\_height and layout\_width properties. Some example values can be “match\_parent” and “wrap\_content”. Match\_parent means that this view is going to match it’s parent’s size, whereas wrap\_content means that this view will be as big as the content inside it requires it to be.
    3. Layout\_gravity and Gravity: These properties define how is the view placed inside its parent view and how is the content inside this view is placed respectively.
    4. Padding and Margin: These properties define how far is the content going to start from inside the view and how far is the view from it’s parent view’s boundary. These concepts are exactly same as in CSS.
    5. Others: There are hundreds of different properties for different kind of components. You can checkout any of these properties at Android developer site.
  1. Viewing the Layout for different configurations:
     1. Landscape and Portrait: A small orientation button in the options menu let’s you view the screen in landscape mode and how all your views will look placed on the screen
     2. Different Screen Types: Output as it would appear on various screen sizes can also be viewed by choosing the desired configuration from the dropdown

1. Menu: Android lets each activity decide its items in the Menu. Activity’s onCreateOptionsMenu is called. You get access to the Menu object in which the menu items need to get added to. Here is example code for adding items defined in main.xml file inside the menu folder.

**public** **boolean** onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.*main*, menu);

**return** **true**;

}

Here is the main.xml file.

<menu xmlns:android=*"http://schemas.android.com/apk/res/android"*

xmlns:app=*"http://schemas.android.com/apk/res-auto"*

xmlns:tools=*"http://schemas.android.com/tools"*

tools:context=*"com.example.listview.MainActivity"* >

<item

android:id=*"@+id/action\_settings"*

android:orderInCategory=*"100"*

android:title=*"@string/action\_settings"*

app:showAsAction=*"never"*/>

<item

android:id=*"@+id/action\_contact\_us"*

android:orderInCategory=*"2"*

android:title=*"Contact Us"*

app:showAsAction=*"never"*/>

</menu>

When one of the menu items is clicked, Android calls activity’s onOptionsItemSelected function.

1. Accessing UI Elements from Java code and manipulating them: Most of the applications don’t just show static data. They accept user input (touch, keyboard etc) and in response change some of the elements in their UI. In order to achieve this we would need to change these elements at Run time.
   1. findViewById: You can use findViewById function to get handle to an element currently available in the view hierarchy. E.g. to access a TextView(id = helloWorld) currently on screen you can use following code.

TextView helloView = (TextView)findViewById(R.id.helloWorld);

* 1. Changing Properties: Once you get the TextView object, you can set all its properties and change its behavior on run time.

E.g. to change its text to contain “Bye World” you can use:

helloView.setText(“Bye World”);

Another Example: Lets change TextView’s alpha

helloView.setAlpha((**float**) 0.4);

1. Event Listeners: UI elements are not just for the appearance we would like to get notified when user tries to interact with these elements. Android achieves this by calling specific function of an assigned object. You can either set it up in the XML file or via getting the handle to the UI elements in the Java file. For further discussion lets assume that the UI element we are talking about is a **button**. Here are some of the events we can listen to:
   1. onClick
      1. Via XML: You can set onClick property of any view inside the xml file. Code for that looks like this:

android:onClick=*"onclick"*

This leads to onclick function of the activity getting called everytime user clicks the view for which this line has been added. Prototype for onclick function must be like this:

**public** **void** onclick(View view)

* + 1. Via Java: We can get access to the view using the findViewById function and set onClickListener. Objects of a class can be set as onClickListener only if that class implements OnClickListener interface defined in android.view.View. Code looks like this:

Button button = (Button) findViewById(R.id.*button1*);

button.setOnClickListener(**this**);

For the above code to compile the current class (MainActivity) must implement the OnClickListener interface.

* 1. onLongClick: You can set onLongClickListener only via Java code. Its not possible to do it via XML. Here is example code:

Button button = (Button) findViewById(R.id.*button1*);

button.setOnLongClickListener(**this**);

For the above code to compile the current class (MainActivity) must implement the OnLongClickListener interface.

* 1. onTouch: onTouchListener also can only be set via Java code. Here is the example code.

Button button = (Button) findViewById(R.id.*button1*);

button.setOnTouchListener(this);

For the above code to compile the current class (MainActivity) must implement the OnTouchListener interface.

There are a lot more event listeners like onKey, onFocusChange etc depending on the UI element. The work exactly like the way onLongClick and onTouch work.

**Exercise**: Add two buttons in the layout and on long click of one button enable/disable the other button. On click of the second button change text of a textfield, second button should be clickable only if the button is enabled.

1. Viewgroups & Layouts: A ViewGroup is a subclass of View class and is a special type of view, which can have children views. Most of the views are not single element views, they will have multiple subviews like buttons, textviews etc. ViewGroups are used as parent views to hold these subviews. Lets look at some subclasses of ViewGroup that we will be using again and again.
   1. Frame Layout: Framelayouts are mostly used as container objects to display a single item. You can add multiple views as well and arrange them by using layout\_gravity on each of the subviews but we should avoid doing that.
   2. Linear Layout: Arranges its children linearly either horizontally (row) or vertically (column). You can use orientation property in XML or setOrientation function in Java to set whether your linear layout is horizontal or vertical. Along with the height and width properties that we have already discussed you can use layout\_weight property to decide the amount of height/width subviews should get within the parent linear layout. An example linear layout:

<LinearLayout

android:layout\_width=*"0dp"*

android:layout\_height=*"match\_parent"*

android:layout\_width=*"match\_parent"*

android:orientation=*"vertical"* >

<TextView

android:id=*"@+id/textView1"*

android:layout\_height=*"0dp"*

android:layout\_width=*"wrap\_content"*

android:layout\_weight=*"2"*

android:text=*"MSFT"* />

<TextView

android:id=*"@+id/textView2"*

android:layout\_height=*"0dp"*

android:layout\_width=*"wrap\_content"*

android:layout\_weight=*"1"*

android:text=*"Microsoft Cooperation"*/>

</LinearLayout>

In this example we have a vertical linear layout and it is divided into two textviews. The first textview gets 2/3rd space where as second one gets 1/3rd of the space.

* 1. Relative Layout: Linear layouts are good enough for a lot of user interfaces but sometimes you want your UI to be more flexible. In these cases we can use relative layouts. In a relative layout every subview tells its position with respect to its parent and its siblings. An example RelativeLayout:

<RelativeLayout

android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"*>

<TextView

android:id=*"@+id/helloWorld"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:layout\_alignParentLeft=*"true"*

android:layout\_alignParentTop=*"true"*

android:text=*"@string/hello\_world"* />

<Button

android:id=*"@+id/button1"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:layout\_alignParentRight=*"true"*

android:layout\_alignTop=*"@+id/helloWorld"*

android:layout\_marginRight=*"26dp"*

android:onClick=*"onclick"*

android:text=*"Button"* />

</RelativeLayout>

Here we have a textview and a button inside a relative layout. Textview is at top left of the layout and button is aligned at right of parent and top of sibling textview. Button also mentions a margin on right side to keep some distance from the parent boundary.

* 1. Coordinator Layout: They are the recent addition to Android Arsenal and useful when the child views need to coordinate with each other, overlap and change positions. For ex: a snackbar that needs to push up the floating action button or a collapsing action bar.

This requires including the design library as follows:

compile 'com.android.support:design:22.2.0'

<android.support.design.widget.CoordinatorLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<android.support.design.widget.FloatingActionButton

android:id="@+id/fab"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="end|bottom"

android:layout\_margin="16dp"

android:src="@drawable/ic\_done" />

</android.support.design.widget.CoordinatorLayout>

Now when a snackbar is displayed, the floating button shifts upwards.

This can also be used to smoothly implement a collapsing toolbar.

1. Designing For Various Screens:
   1. Values - The basic file, if no other values are provided, values are picked from here.
   2. Values-sw600dp : Used to support 7” inch screens (tablets)
   3. Values-sw720dp: Used to support 10” tablets

All available variants have landscape and portrait modes.

1. Converting from dp to pixels: It is sometimes required to express distance in pixels especially while animating objects and other gesture based events

// The gesture threshold expressed in dp  
private static final float GESTURE\_THRESHOLD\_DP = 16.0f;  
  
// Get the screen's density scale  
final float scale = [getResources()](http://developer.android.com/reference/android/content/ContextWrapper.html#getResources()).[getDisplayMetrics()](http://developer.android.com/reference/android/content/res/Resources.html#getDisplayMetrics()).density;  
// Convert the dps to pixels, based on density scale  
mGestureThreshold = (int) (GESTURE\_THRESHOLD\_DP \* scale + 0.5f);  
  
// Use mGestureThreshold as a distance in pixels...

1. Activity Life Cycle: An activity goes through a complete life cycle of getting created to getting destroyed with many other events in the middle. Android calls a specific method for each of these events. These methods can be used to set up the UI, setup click/touch listeners or doing clean up like closing network connection etc. Here is a list of life cycle methods for an activity.
   1. onCreate: This method is called when the activity is created. This is the place where you should call setContentView and set activity’s view. This is where you should set up list’s data and create other views.
   2. onRestart: This is called when an activity is stopped and restarted instead of getting destroyed. This is always followed by onStart.
   3. OnStart: This method is called when the activity starts becoming visible to the user.
   4. onResume: This is when user can start interacting with the activity. Now this activity is in focus.
   5. onPause: This function is called when the activity is visible but is loosing focus. E.g. if a new activity is being loaded or a AlertDialog shows up in front of the current activity.
   6. onStop: This is called when the activity is no longer visible to the students. This can happen because a new activity is being created or a previous activity is being pushed to the front.
   7. onDestroy: This is the final call you will receive before the activity is destroyed. An activity can be destroyed either because the system is running low on resources and is killing your activity or the activity has finished. The first condition can only happen if your activity is in background.



1. Saving and restoring activity state: As we know that an inactive activity can be destroyed if the system is running low on resources, but the user should be oblivious of the destruction of the activity. In order to achieve this whenever the user tries to go back to the destroyed activity, system will create a new instance and rebuilds the activity. To achieve this Android stores the state data into a **Bundle**, which is a key value store.

Here is how data is stored:

* 1. View Objects: Android will take care of storing all the views in the activity. It will also take care of any data corresponding to the views e.g. you don’t have to worry about the text entered by the user in the edittext field.
  2. Additional data: You need to override the onSaveInstanceState function to save additional data. Android will pass you the Bundle object in which you can add additional data, which you think, might be required to rebuild the activity.

Retrieving Data: You have two options, where you can access the additional data that you stored.

1. onCreate: You get passed the Bundle object in this function if the activity was destroyed and needs to be started from same state. You need to check if the bundle object is null or not because this function gets called even when it’s a fresh start to the activity and no data needs to be restored. You should try to restore data if the Bundle is not null.
2. onRestoreInstanceState: This function gets called only if the activity was destroyed and needs to be restored. Inside this you can directly access the bundle. This will get called after the onStart function.
3. Homework
   1. Add a button and on click of a button show a Toast message to user. Read about Toasts.
   2. Add a textview and show count of button clicks in the textview.
   3. Add [logging](http://developer.android.com/reference/android/util/Log.html) to all the lifecycle functions and check the order in which they get called.
   4. Add code to store and restore the button click count.
      1. You can test it by enabling the “Don’t keep activities” option in the developer settings.
   5. Build a basic calculator