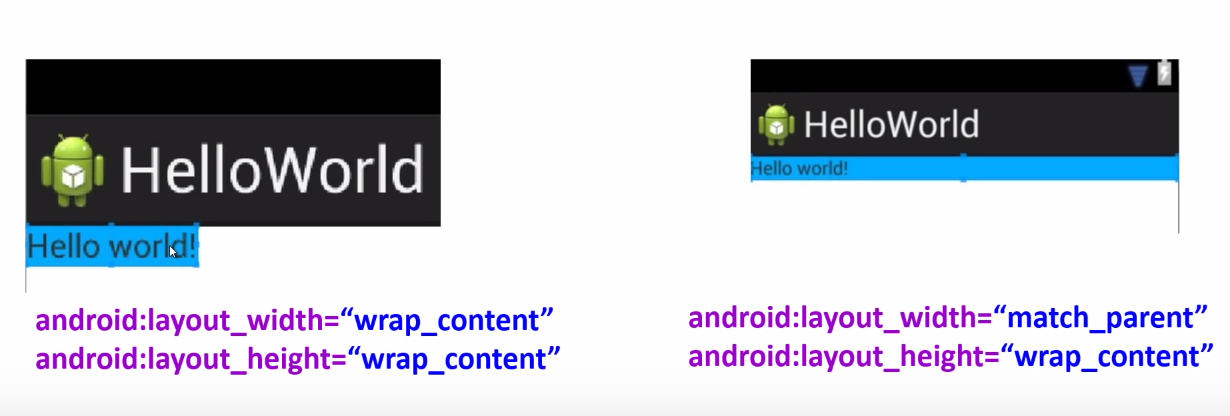
Android app consists of the following things:

1. Java code files
2. Resources
   1. drawable (images + other stuff, icons)
   2. layout (appearance: xml files, horizontal/vertical)
      1. texts/buttons are called views in android
      2. combination of views if called a viewgroup
      3. a viewgroup may have a viewgroup inside
      4. wrap\_content: Take only as much you need
      5. match\_parent: Take everything you can



* 1. values (texts)
     1. strings.xml (“Hello World”)

<resources>

<string name = “hello\_world”> Hello World <string>

</resources>

* 1. others (xml files)

1. Configuration files

Activity: An application component that provides a screen. Every app has 1 main activity.

* When a new activity starts, the previous activity is thrown in the back stack.
* Lifecycle methods – tells if the activity (app) is visible to the user/paused/stopped.
* Callback Methods – Android OS calls certain methods on our activity to notify whether our app is running or not.

Activity Lifecycle Methods:

* When the user clicks on the app, following methods are called in quick succession.
  + - onCreate(),
    - onStart(),
    - onResume()
* When the user pauses/minimizes the app, following methods are called.
  + - onPause(),
    - onStop()
* If we come back to the app again after it was paused following methods are called.
  + - onRestart()
    - onStart()
    - onResume()
* When the app is killed, onDestroy() method is called.
* When the orientation is changed, everything is destroyed. A fresh start is made. This happens because the entire orientation changes and whatever is drawn on the screen has to be re-drawn.

Logcat – Used for debugging purposes. Prints different messages to the logcat terminal.

Log.d(string tag, string message)

In order to make sure that the onConfigurationChanged method is called when the screen is rotated, add screenSize to the configChanges in AndroidManifest.xml because an orientation change would also imply a screen size change.

Whenever an id is given to a control in android, it saves the data when the orientation is changed.

Activity object is always held in the memory when the app is paused or stopped, unless the system explicitly destroys it.

* Use onSaveInstanceState() to save information
* onSaveInstanceState(Bundle out) { out.put(..); } out is a map
* Call super.onSaveInstanceState() to save View data in your activity
* onRestoreInstanceState() : Once the system recreates activity after destroying it, onRestoreInstanceState() and onCreate() receive the same Bundle.
* Cannot restore state after back button is pressed.
* onCreate() has the same Bundle as onRestoreInstanceState() as when the screen is rotated, the activity is destroyed and a new start is made with the same data which was there before rotating the screen.
* Call super.onRestoreInstanceState() to restore view data in your activity.

Intents:

* Intents are used to start services.
* Explicit Intents: When you know the name of the guy you're calling
* Implicit Intents: Don’t know who to call. (don’t know who is going to do the functionality for you)
* Activity A ----- (Intent) -----> Activity B
* Ways to create an intent:
  + Intent I = new Intent(Context c, Class d)
  + startActivity(I)
  + or, Intent I = new Intent()
  + I.setClassName(string packageName, string className)
  + startActivity(I)

Whenever you want to handle implicit intents, you must declare an intent-filter. When we send an intent, the intent gets compared on the receiving side by the intent filter to check if it matches with an activity/service/broadcast receiver and that corresponding activity/service/broadcast receiver is called.

Intent Filters :

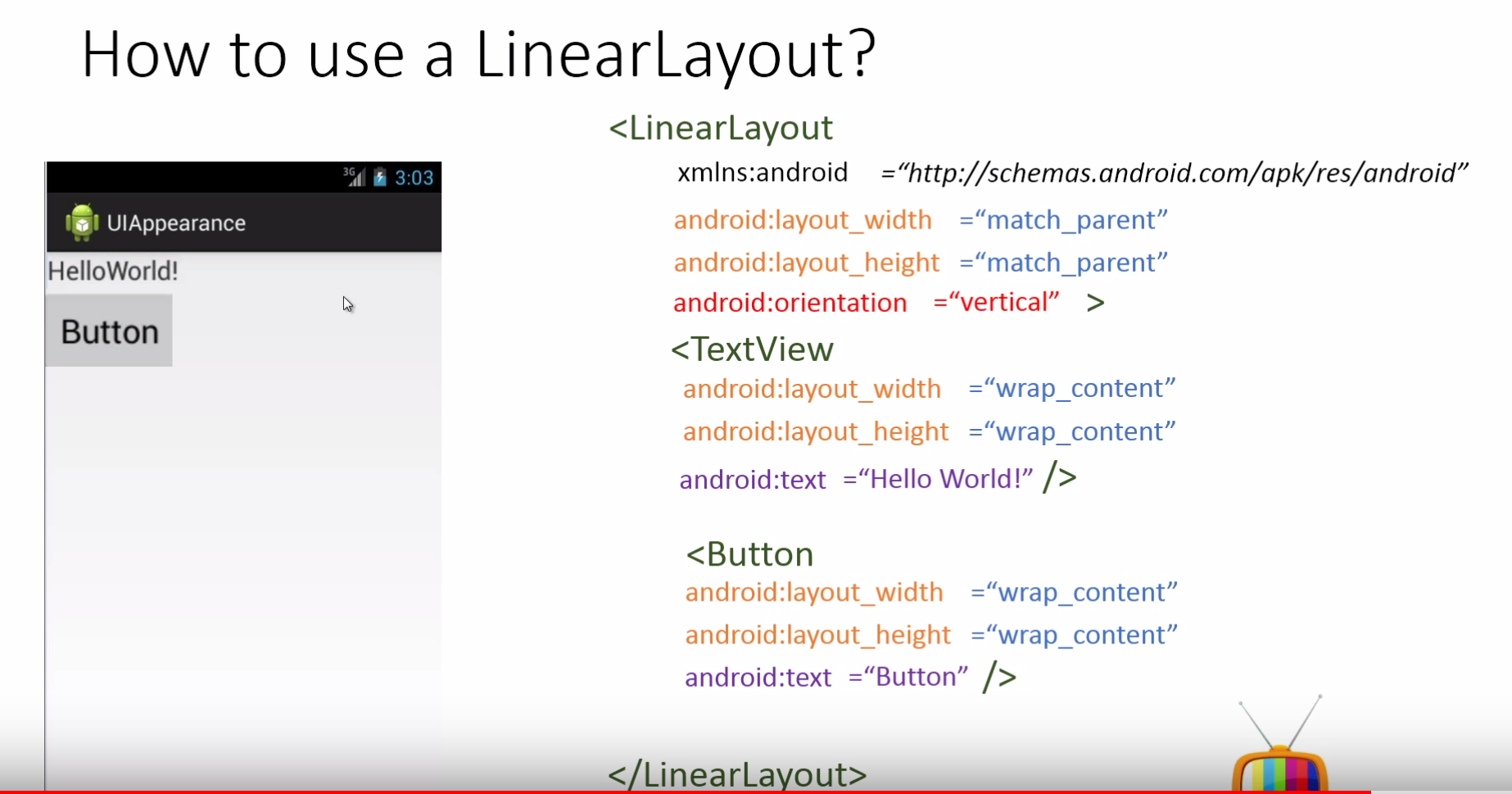
* Describe what ACTION,DATA,CATEGORIES a component can handle
* Defined mostly in manifest file but can also be registered in the code.
* What do you write inside intent-filter ?
  + One or more action to indicate what action can be handled by your component
  + One or more category to indicate what grouping does your component support
  + Data, to indicate what data can be handled by your component

How to create a Custom Toast:

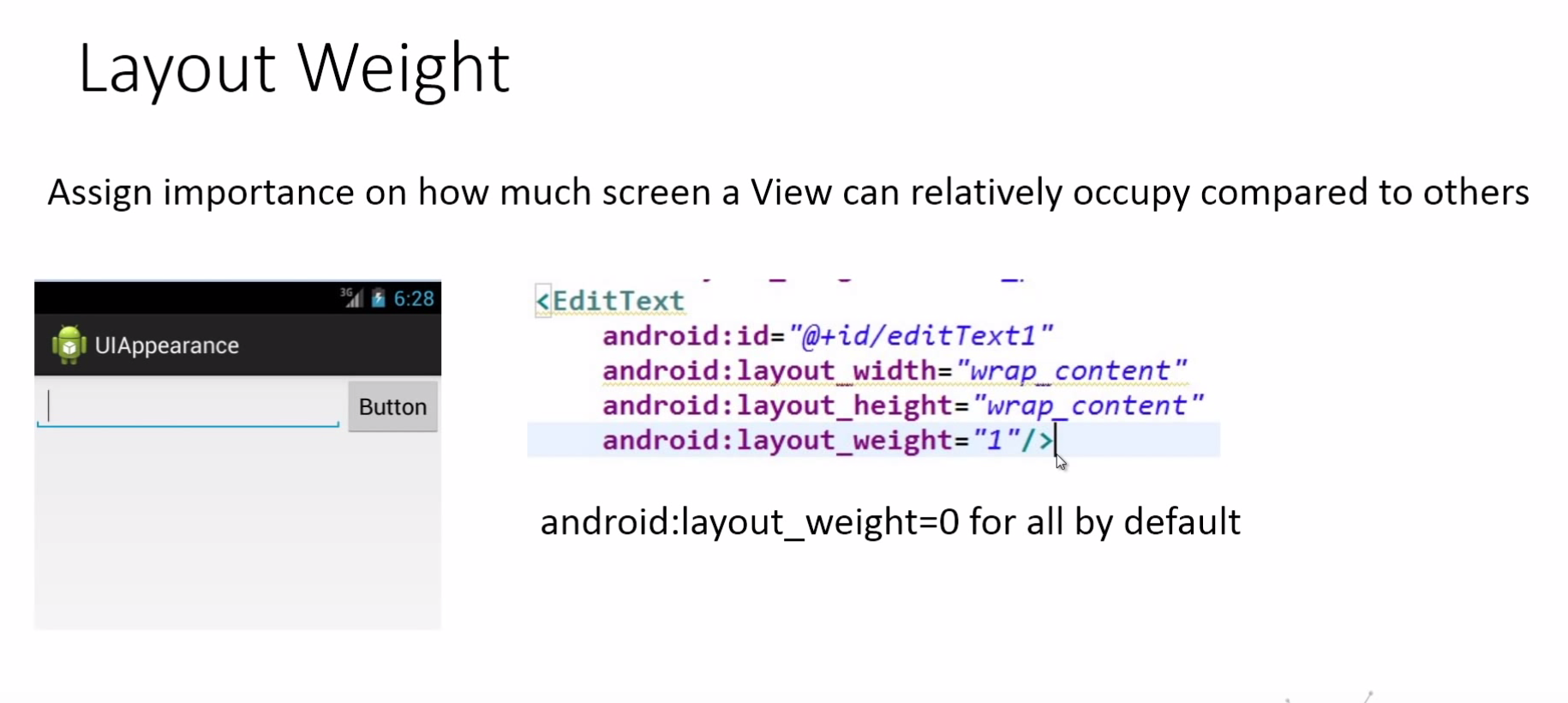
* Create a layout file that displays what your toast should look like
* <LinearLayout> <ImageView/> <TextView/> </LinearLayout>
* Create a simple toast using the following constructor
  + Toast toast = new Toast(Context c);
  + toast.setDuration(Toast.LENGTH\_SHORT);
  + toast.setGravity(Gravity.CENTER, 0, 0);
* Construct a view from xml that will be used as Toast's view (need to convert xml appearance to a java object). LayoutInflater is a class that reads xml based description and converts it into a java View object.
  + LayoutInflater lin = getLayoutInflater();
  + lin.inflate(int, ViewGroup); int refers to the id, Viewgroup refers to the parent in the hierarchy.
  + View appear = lin.inflate(R.layout.toast\_layout, (ViewGroup) findViewById(R.id.toast\_linear\_layout))
  + toast.setView(appear);
  + toast.show()

What is a View: An object that takes rectangular area on the screen and is responsible for handling events in that area.

What is a ViewGroup: An object that surrounds a View and is invisible and is responsible for holding views in a certain fashion.



Layout Weight: Assign importance on how much screen a view can relatively occupy as compared to others.

The button takes up wrap\_content, so when layout\_weight = 1, it tells the EditText to take up all the space which is left after giving wrap\_content to the button.

