ANDROID REVIEW

Ho Dac Hung

VIEW (CLASS)

- View class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components

VIEW (CLASS)

- The ViewGroup subclass is the base class for layouts, which are invisible containers that hold other Views (or other ViewGroups) and define their layout properties.

VIEW (USING)

- All of the views in a window are arranged in a single tree. You can add views either from code or by specifying a tree of views in one or more XML layout files. There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other content.

VIEW (USING)

- Set properties
- Set focus
- Set up listeners
- Set visibility

VIEW (CUSTOM VIEW)

- Creation
- Layout
- Drawing
- Event processing
- Focus
- Attaching

VIEW (IDs)

- Views may have an integer id associated with them. These ids are typically assigned in the layout XML files, and are used to find specific views within the view tree. A common pattern is to: define a Button in the layout file and assign it a unique ID; from the onCreate method of an Activity, find the Button.

VIEW (POSITION)

- The geometry of a view is that of a rectangle. A view has a location, expressed as a pair of left and top coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel.

SURFACE VIEW

- Provides a dedicated drawing surface embedded inside of a view hierarchy. You can control the format of this surface and, if you like, its size; the SurfaceView takes care of placing the surface at the correct location on the screen.

LAYOUTS

- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways: Declare UI elements in XML, Instantiate layout elements at runtime.

LAYOUTS (WRITE THE XML)

- Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML — with a series of nested elements.
- Each layout file must contain exactly one root element, which must be a View or ViewGroup object. Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.

LAYOUTS (LOAD THE XML)

- When you compile your application, each XML layout file is compiled into a View resource. You should load the layout resource from your application code, in your Activity.onCreate() callback implementation. Do so by calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.layout_file_name.

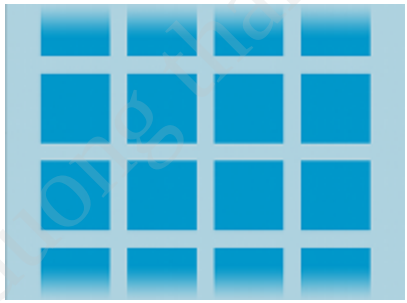
LAYOUTS (LINEAR)

LAYOUTS (LINEAR)

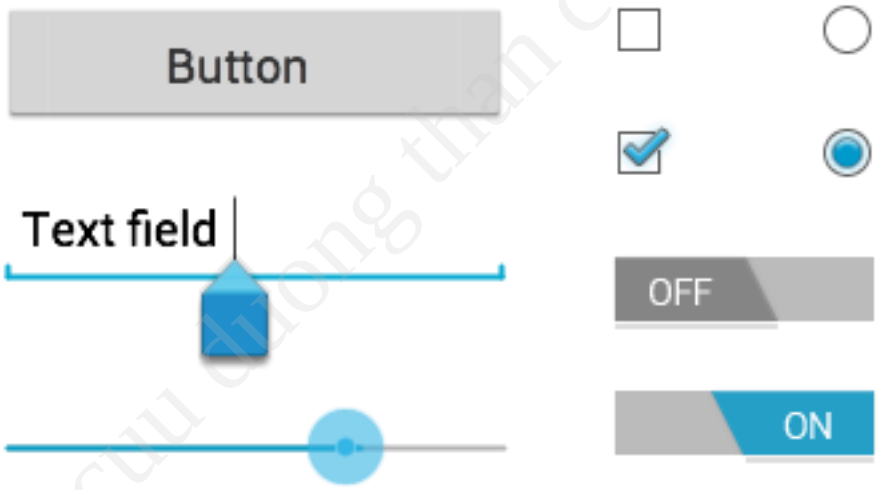
LAYOUTS (WEB)

LAYOUTS (LIST)

LAYOUTS (GRID)



INPUT CONTROLS



INPUT CONTROLS

- Button
- Text field
- Checkbox
- Radio button
- Toogle button
- Spinner
- Pickers

INPUT EVENTS

- On Android, there's more than one way to intercept the events from a user's interaction with your application. When considering events within your user interface, the approach is to capture the events from the specific View object that the user interacts with. The View class provides the means to do so.

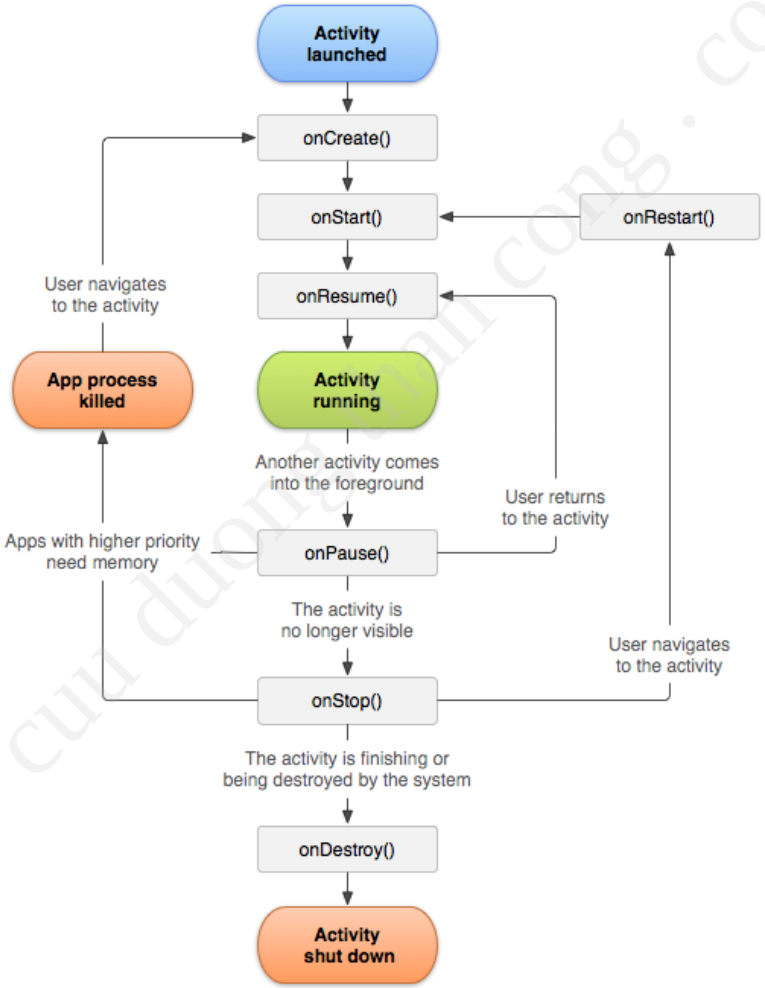
INPUT EVENTS

- onClick()
- onLongClick()
- onFocusChange()
- onKey()
- onTouch()
- onCreateContextMenu()

ACTIVITY

- An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View).

ACTIVITY LIFECYCLE



ACTIVITY LIFECYCLE

- protected void onCreate(Bundle savedInstanceState);
- protected void onStart();
- protected void onRestart();
- protected void onResume();
- protected void onPause();
- protected void onStop();
- protected void onDestroy();

STARTING ACTIVITYS

- The startActivity(Intent) method is used to start a new activity, which will be placed at the top of the activity stack. It takes a single argument, an Intent, which describes the activity to be executed.

GETTING RESULTS

- Sometimes you want to get a result back from an activity when it ends. For example, you may start an activity that lets the user pick a person in a list of contacts; when it ends, it returns the person that was selected. To do this, you call the startActivityForResult(Intent, int) version with a second integer parameter identifying the call. The result will come back through your onActivityResult(int, int, Intent) method.

SAVING PERSISTENT STATE

- There are generally two kinds of persistent state than an activity will deal with: shared document-like data (typically stored in a SQLite database using a content provider) and internal state such as user preferences.

SAVING PERSISTENT STATE

- Shared document-like data
- Internal state

PERMISSIONS

- The ability to start a particular Activity can be enforced when it is declared in its manifest's <activity> tag. By doing so, other applications will need to declare a corresponding <uses-permission> element in their own manifest to be able to start that activity.

PROCESS LIFECYCLE

- The Android system attempts to keep application process around for as long as possible, but eventually will need to remove old processes when memory runs low. As described in Activity Lifecycle, the decision about which process to remove is intimately tied to the state of the user's interaction with it. In general, there are four states a process can be in based on the activities running in it, listed here in order of importance. The system will kill less important processes (the last ones) before it resorts to killing more important processes (the first ones).

APP MANIFEST

- Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.