Layout

- Layout is the architecture for the user interface in an Activity (or actually in any element that has a view, more on this later). It defines the view structure and holds all the elements (views and view groups) that are visible to the user.
- Layout must contain exactly one root element, which must be a View or, more often, ViewGroup object (or inheriting from them). 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 can be declared in two ways:
  - Declare UI elements in XML - this is the recommended way
  - Instantiate layout elements at runtime in your Java code

Layout / XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</LinearLayout>
```

Layout / Java

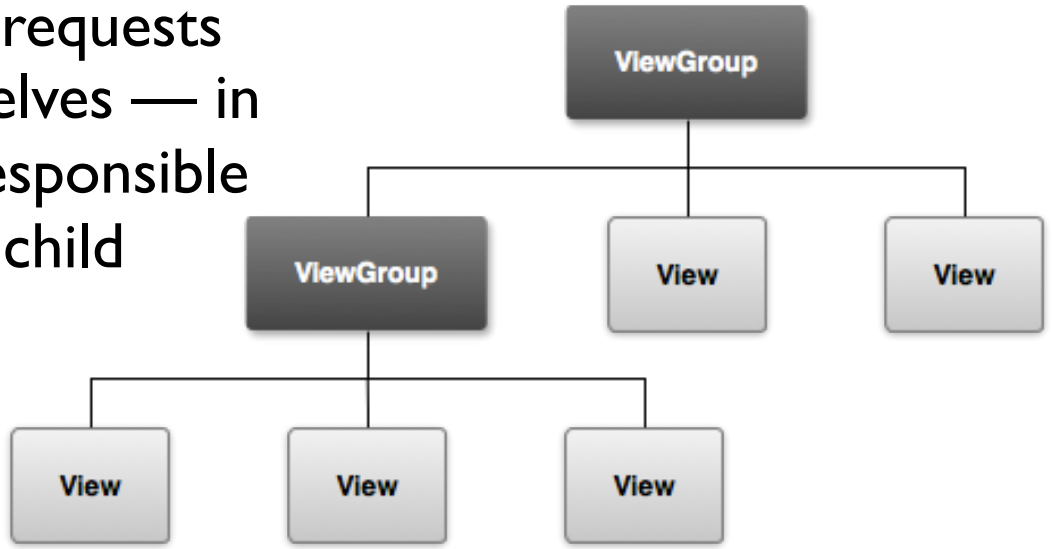
```
    protected void onCreate(Bundle savedInstanceState) {
        Log.d(TAG, "onCreate()");

        super.onCreate(savedInstanceState);

        layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);

        TextView tv = new TextView(this);
        tv.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
                                            LayoutParams.WRAP_CONTENT));
        tv.setText("Hello world");
        layout.addView(tv);

        tv = new TextView(this);
        tv.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
                                            LayoutParams.WRAP_CONTENT));
        tv.setText("Goodbye world");
        layout.addView(tv);

        setContentView(layout);
    }
```

View

- A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen.
- A View object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides.
- A View is also a point of interaction for the user and the receiver of the interaction events.
- Android provides a set of fully implemented widgets (ready-made view objects), like buttons, checkboxes, and text-entry fields. Some widgets provided by Android are more complex, like a date picker, a clock, and zoom controls.

View hierarchy

- In order to attach the view hierarchy tree to the screen for rendering, an Activity must call the `setContentView()` method and pass a reference to the root node object.
- The Android system receives this reference and uses it to instantiate, measure and draw the tree.
- The root node of the hierarchy requests that its child nodes draw themselves — in turn, each view group node is responsible for calling upon each of its own child views to draw themselves.



Some ViewGroup subclasses

- **FrameLayout**
  - It's a blank space on your screen that you can later fill with a single object — for example, a picture that you'll swap in and out.
  - All child elements of the FrameLayout are pinned to the top left corner of the screen. Thus subsequent child views will simply be drawn over previous ones, partially or totally obscuring them (unless the newer object is transparent).
- **LinearLayout**
  - Aligns all children in a single direction — vertically or horizontally, depending on the orientation attribute.
  - All children are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding).
  - A LinearLayout respects margins between children and the gravity (right, center, or left alignment) of each child.
  - Relative sizes of elements in LinearLayout can be controlled with layout_weight parameter. LinearLayout real estate is assigned to child views based on their layout_weight relative to sum of all layout_weight values in the layout.

Some ViewGroup subclasses

- **TableLayout**
  - Positions its children into rows and columns.
  - TableLayout containers do not display border lines for their rows, columns, or cells.
  - The table will have as many columns as the row with the most cells.
  - A table can leave cells empty, but cells cannot span columns, as they can in HTML.
- **RelativeLayout**
  - Lets child views specify their position relative to the parent view or to each other (specified by ID).
  - Elements are rendered in the order given, so if the first element is centered in the screen, other elements aligning themselves to that element will be aligned relative to screen center.
  - Also, because of this ordering, if using XML to specify this layout, the element that you will reference (in order to position other view objects) must be listed in the XML file before you refer to it from the other views via its reference ID.

Some UI widgets

- TextView 

- EditText 

- Button 

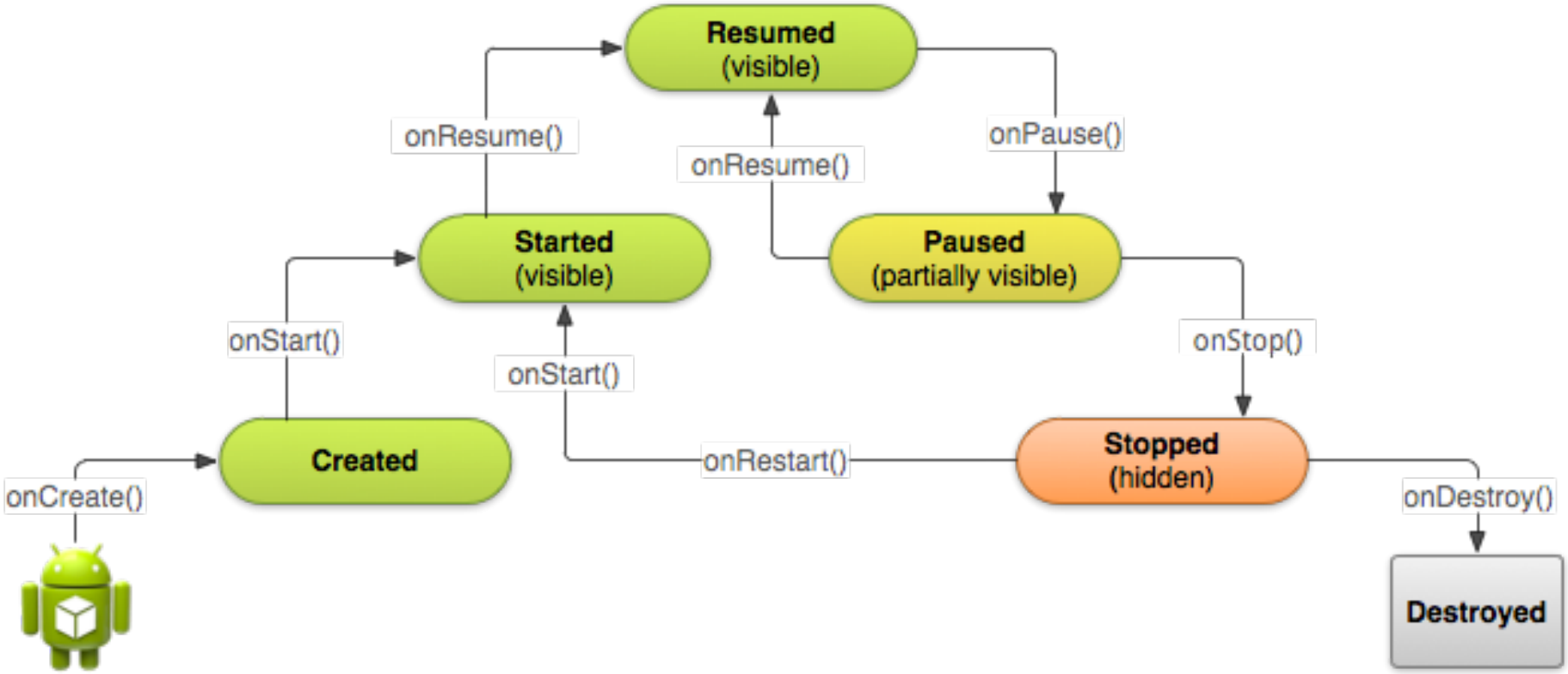
- ImageButton
- CheckBox 

- ToggleButton 

- RadioButton
- RadioGroup 

- See http://developer.android.com/guide/topics/ui/controls.html

Activity Lifecycle



- http://developer.android.com/training/basics/activity-lifecycle/starting.html

Reading list

- http://developer.android.com/guide/topics/ui/declaring-layout.html (you can skip material starting from Building Layouts with an Adapter)
- http://developer.android.com/guide/topics/ui/ui-events.html
- http://developer.android.com/guide/components/activities.html