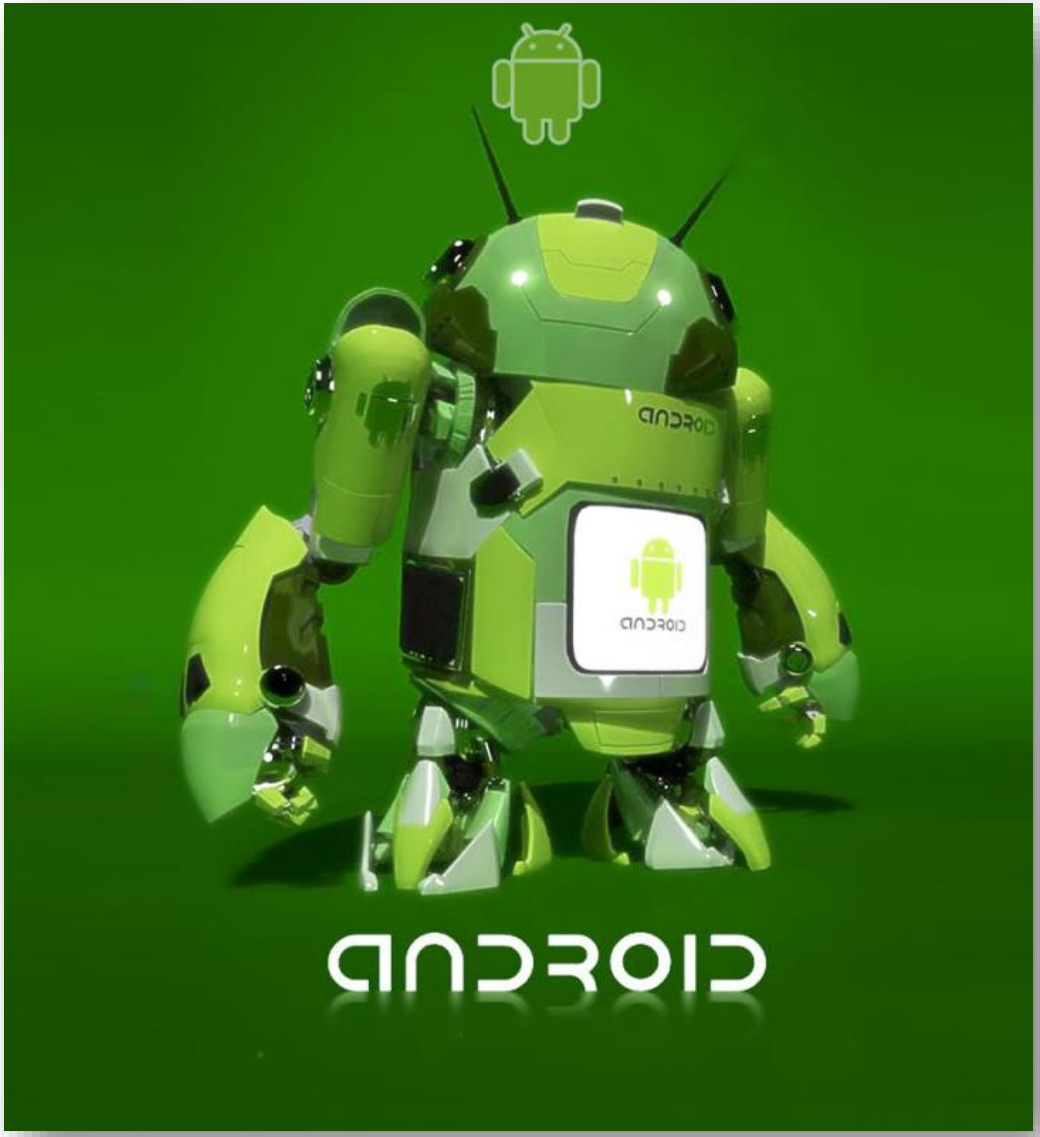
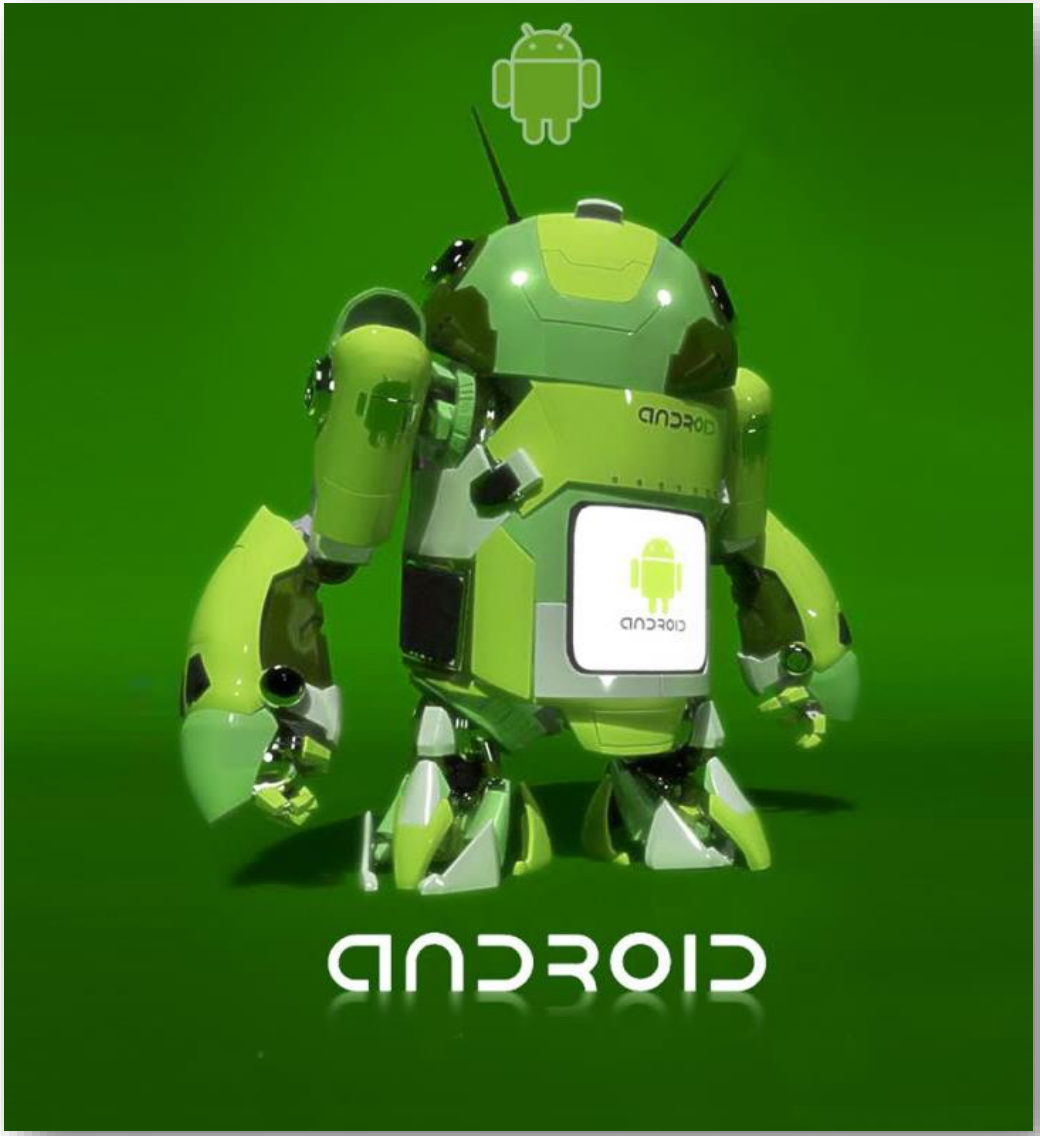
Week 4

MONASH University

- Views, ViewGroups, Layouts, Layout.Params
- Creating UIs in XML and Java
- Layout Types, ConstraintLayout
- Calculator Apps
- Styles, Themes, Material Design
- Action Bar vs. App Bar



- Views, ViewGroups, Layouts, Layout.Params



Views, ViewGroups, Layouts, LayoutParams

- Android UIs
  - Are made up of a hierarchy of View objects
- View
  - Package: android.view.View
  - "This 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 (buttons, text fields, etc.). 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."
- ViewGroup
  - Package: android.view.ViewGroup
  - "A ViewGroup is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers. This class also defines [nests] the ViewGroup.LayoutParams class which serves as the base class for layouts parameters."

See: https://developer.android.com/reference/packages.html

Source: https://developer.android.com/reference/android/view/View.html

See: https://developer.android.com/reference/packages.html

Source: https://developer.android.com/reference/android/view/ViewGroup.html

ViewGroup, ViewGroup.LayoutParams

- ViewGroups
  - Allow Views to be nested which means they can be represented as a hierarchy (upside down tree)
    - Think HTML and XML
  - This tree, in an Android UI, is usually headed by a special kind of ViewGroup called a Layout

A Layout is responsible for managing the size, position and behaviour of all the Views (e.g. widgets) it contains

- ViewGroup.LayoutParams
  - "LayoutParams are used by views to tell their parents how they want to be laid out. See ViewGroup Layout Attributes for a list of all child view attributes that this class supports.
  - The base LayoutParams class just describes how big the view wants to be for both width and height. For each dimension, it can specify one of:
    - FILL_PARENT (renamed MATCH_PARENT in API Level 8 and higher), which means that the view wants to be as big as its parent (minus padding)
    - WRAP_CONTENT, which means that the view wants to be just big enough to enclose its content (plus padding)
    - an exact number
  - There are subclasses of LayoutParams for different subclasses of ViewGroup. For example, AbsoluteLayout has its own subclass of LayoutParams which adds an X and Y value."

Source: https://developer.android.com/reference/android/view/ViewGroup.LayoutParams.html

LayoutParams

Source: https://developer.android.com/guide/topics/ui/declaring-layout.html

- Layout Parameters
  - "XML layout attributes named layout_something define layout parameters for the View that are appropriate for the ViewGroup in which it resides.
  - Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains property types that define the size and position for each child view, as appropriate for the view group. As you can see in figure 1, the parent view group defines layout parameters for each child view (including the child view group)."

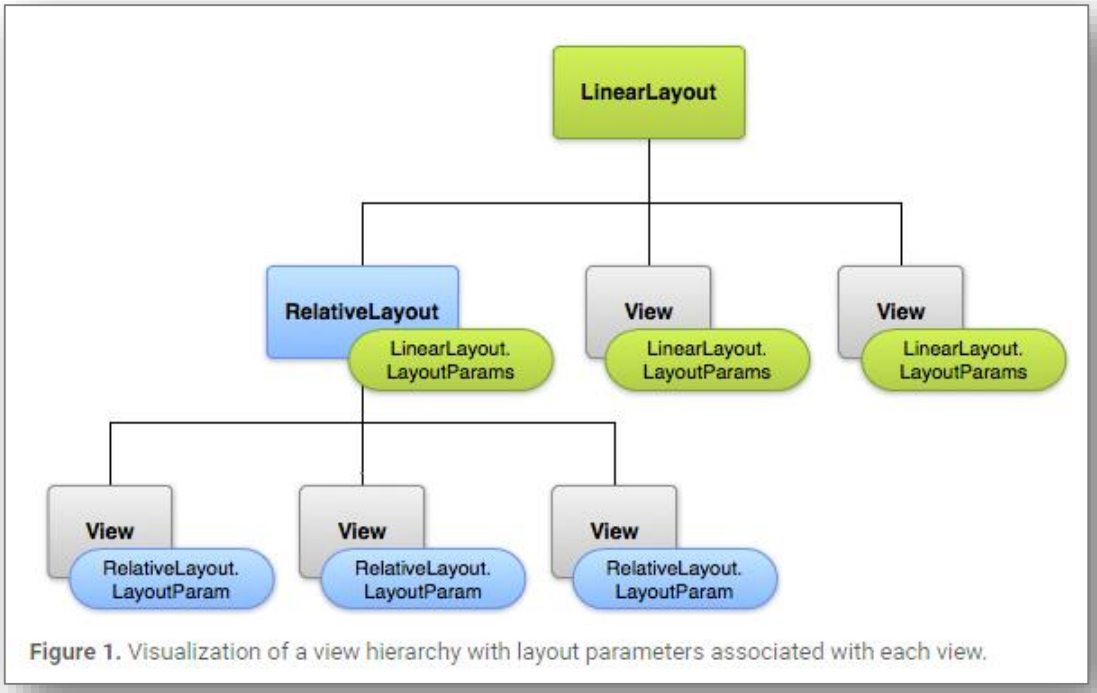
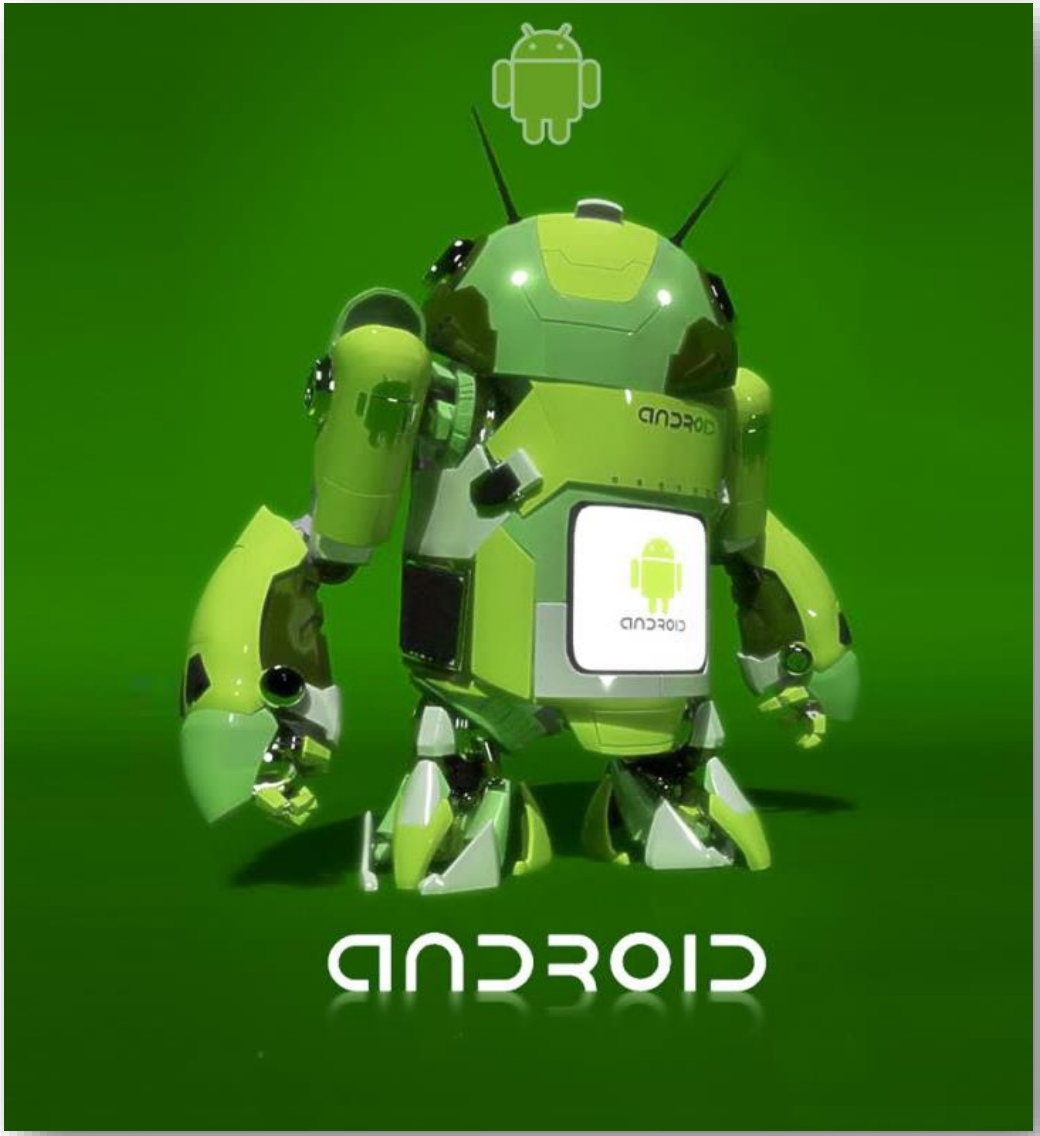


Figure 1. Visualization of a view hierarchy with layout parameters associated with each view.

"These [layout] attributes are specified with the rest of a view's normal attributes (such as background, but will be parsed by the view's parent and ignored by the child."

Source: here

- Creating UIs in XML and Java



Creating UIs

Note: “layout” (small l) in this quote refers to the entire UI not just the ViewGroup that encloses it.

- 2 Ways to create a UI
  - “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. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
    - Instantiate layout elements at runtime. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.
  - The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI. For example, you could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties. You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.”

Source: https://developer.android.com/guide/topics/ui/declaring-layout.html

Java and XML

- Java or XML?

Source: https://developer.android.com/guide/topics/ui/declaring-layout.html

  - "The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior. Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile. For example, you can create XML layouts for different screen orientations, different device screen sizes, and different languages. Additionally, declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems. As such, this document focuses on teaching you how to declare your layout in XML. If you're interested in instantiating View objects at runtime, refer to the ViewGroup and View class references."
- Java ←→ XML
  - "In general, the XML vocabulary for declaring UI elements closely follows the structure and naming of the classes and methods, where element names correspond to class names and attribute names correspond to methods. In fact, the correspondence is often so direct that you can guess what XML attribute corresponds to a class method, or guess what class corresponds to a given XML element. However, note that not all vocabulary is identical. In some cases, there are slight naming differences. For example, the EditText element has a text attribute that corresponds to EditText.setText()."

Java and XML

- How to create a UI in Java (no XML required)
  - Check out the JavaLayout app
    - Downloadable from Moodle
    - Note the difference between how widget properties (internal to the widget) and layout parameters (specified on widget but processed by parent) are set
- Basic Procedure
  - For each widget
    - Instantiate and customise by setting its properties usually using setPropName methods
    - Instantiate a LayoutParams object for the widget
      - Type depends on containing layout (e.g. RelativeLayout.LayoutParms)
      - Use constructor and addRule method to specify the size, position and behaviour of the widget in the containing layout
    - Instantiate the Layout of the required type
    - Use Layout's addView method to add widget (as 1st parameter of addView) and specify its associated LayoutParams object (as a 2nd parameter of addView)

LayoutParams

- View Properties and LayoutParams
  - Properties
    - Belong to the View object's Class
    - e.g. TextViews have a setPadding methods which sets a TextView's padding (something internal to the View)
  - LayoutParams
    - Belong to the Class of the Layout a View is contained by
    - e.g. all LayoutParam classes that are subclasses of MarginLayoutParam (itself a direct subclass of ViewGroup.LayoutParams) have a method setMargins which sets a contained View's margins (a relationship between views contained by the same container)
- Look the same as properties in XML
  - Except their names start like android:layout_
- Are handled very differently in Java code
  - See previous and next slide

Java and XML

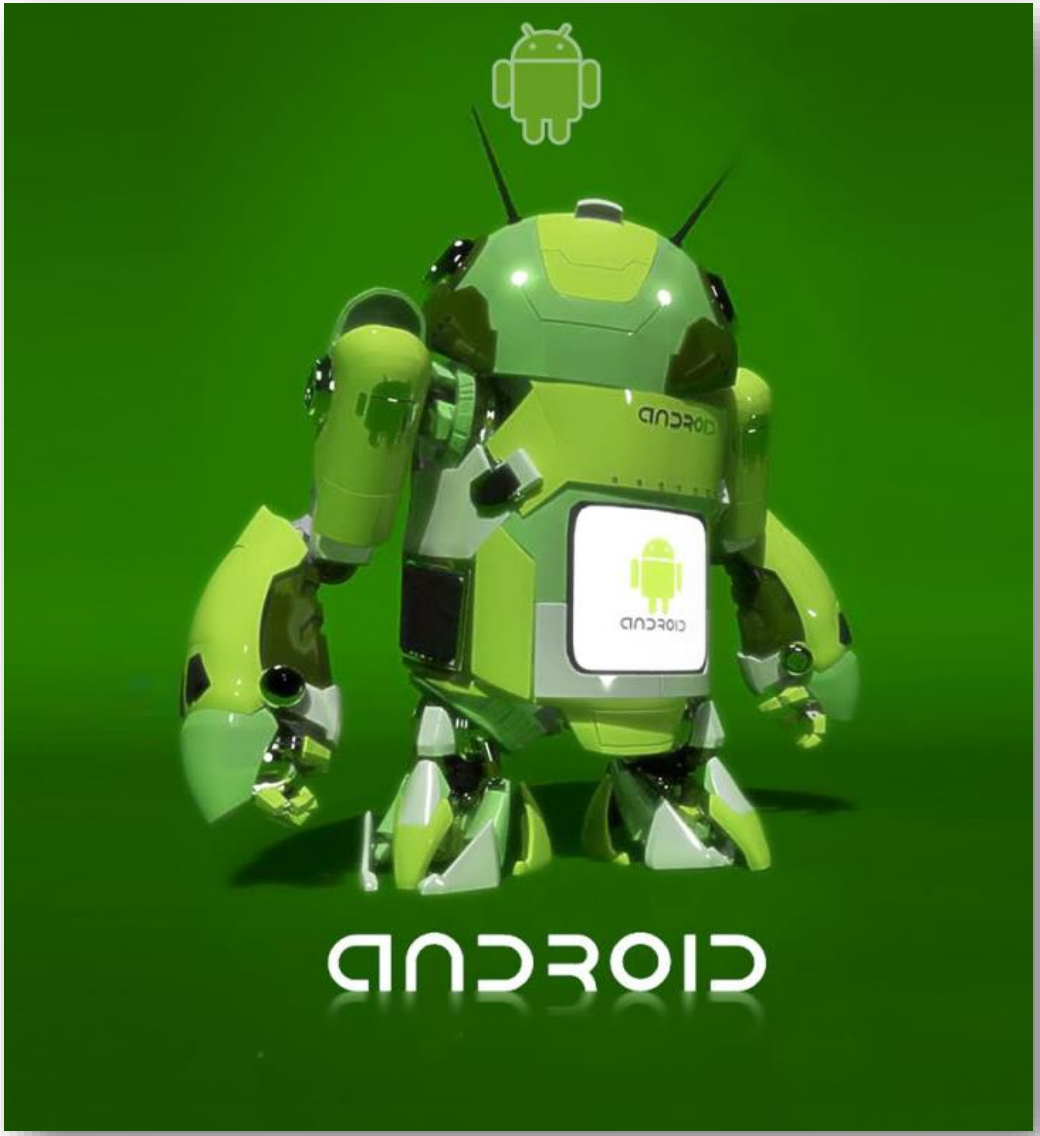
* e.g. TextView: attribute android:hint,
method setHint(…) and getHint()

- Correspondence between Java and XML – Example
  - Check out JavaVsXML
    - Downloadable from Moodle
  - View Properties are often straightforward
    - XML: widget attribute: android:text="…"
    - Java: widget method : .setText(…);
    - Conversion from dp/sp to pixels is required in code for some method parameters
  - View Layout parameters are a bit complicated in code
    - e.g. the layout instruction is a rule (boolean or refers to sibling)
      - XML: widget attribute: android:layout_centerHorizontal="true"
      - Java: Rule added to widgets's LayoutParams: .addRule(RelativeLayout.CENTER_HORIZONTAL)
    - e.g. the layout instruction is not a rule (i.e. requires parameters)
      - XML: widget attribute: android:layout_marginBottom="20dp"
      - Java: widget method: setMargins(0, 0, 0, dpToPxConversion)
      - Note: RelativeLayout.LayoutParams inherits setMargins from its direct super class ViewGroup.MarginLayoutParams

View Doco – Properties and LayoutParams

- View Properties
  - Go to the View's API class reference page (e.g. TextView)
    - Use the search box on https://developer.android.com/index.html
    - Remember such pages are topped by summary tables with links to more details way below
  - Table of XML attributes (beware inherited XML attributes)
    - This is for attribute equivalents to View properties NOT View LayoutParams
    - Beware ctrl + f will not work if the attribute is inherited (e.g. onClick)
      - Not until you expand the inherited section and refresh the ctrl + f search box
  - Table of Java Methods (beware inherited Java methods)
    - To set and get a View's property values
- View Layout Params
  - Depends on which type of Layout the View will be contained by
  - Go to the relevant LayoutParams page (e.g. RelativeLayout.LayoutParams)
  - Table of XML attributes (beware inherited XML attributes)
    - Use as XML attributes or in Java code add to the View's associated LayoutParams object using that objects addRule method
  - Table of Java Methods (beware inherited Java methods)
    - Apply directly to the View's associated LayoutParms object

- Layout Types,
- ConstraintLayout



Layouts Types

- ViewGroup
  - Has several Layout direct subclasses
    - e.g. CoordinatorLayout, FrameLayout, GridLayout, LinearLayout, RelativeLayout
  - Has several Layout indirect subclasses
    - e.g. TableLayout
  - Which Layout type to use depends on the requirements of the UI you are trying to create

    See: https://www.tutorialspoint.com/android/android_user_interface_layouts.htm
  - Don't forget since a ViewGroup is a View a ViewGroup can be contained by another ViewGroup
    - So Layouts can be nested to optimise each part of your UI
    - Google strongly recommends against nesting deeply for efficiency and clarity reasons
      - This is one reason Google have introduced ConstraintLayout (see below)
- In addition: View containers
  - There are many direct and indirect View container subclasses that can be part of a UI's View hierarchy (e.g. Toolbar)
- In addition: ConstraintLayout
  - A new direct subclass found in the support library
    - Use SDK Manager to download the Android Support Repository, specifically, the "ConstraintLayout for Android" and the "Solver for ConstraintLayout" libraries
    - Edit Gradle scripts to include the necessary dependencies

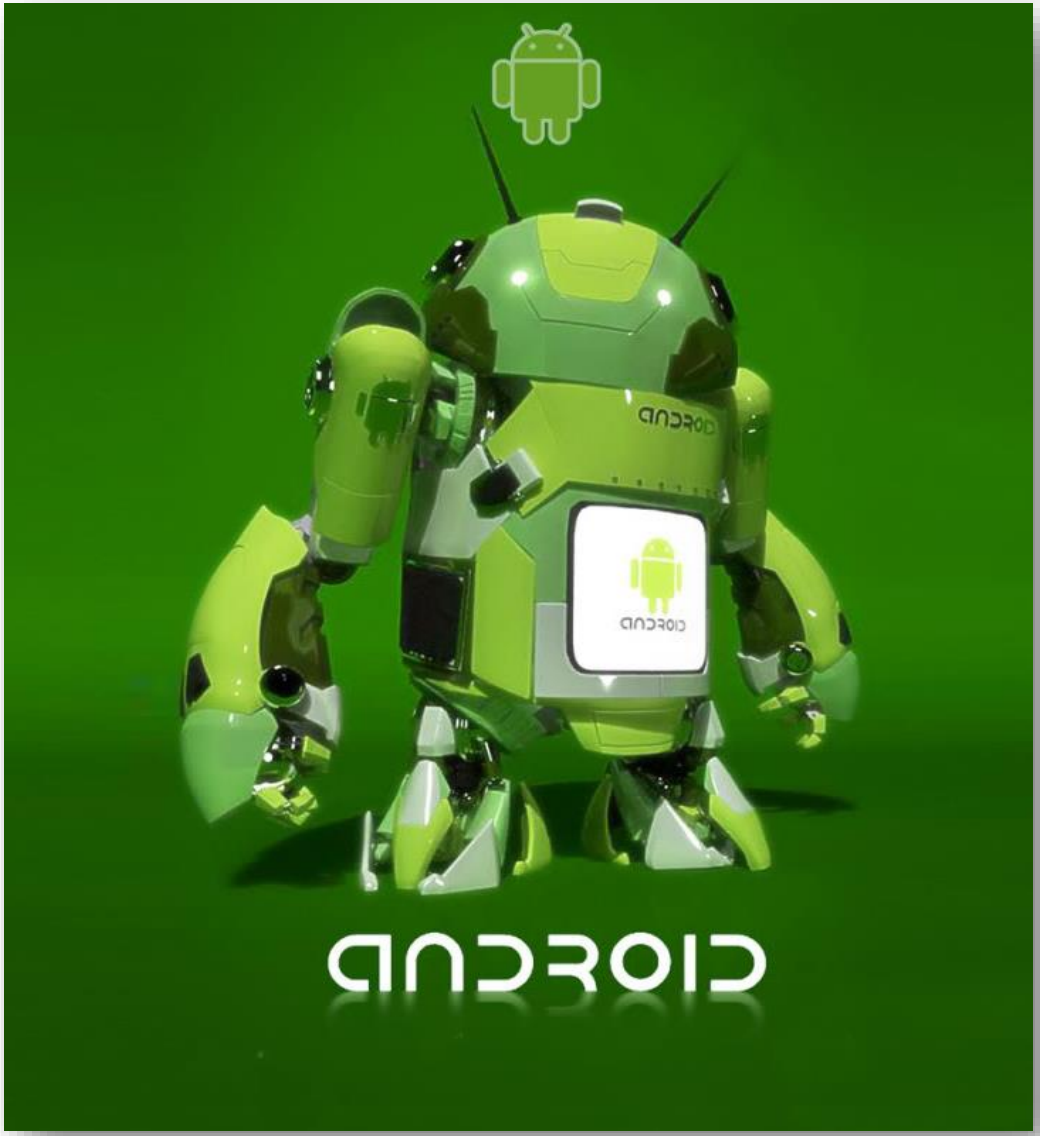
ConstraintLayout

- Requirements
  - “ConstraintLayout is available in an API library that's compatible with Android 2.3 (API level 9) and higher, and the new layout editor is available in Android Studio 2.2 and higher”
- ConstraintLayout.LayoutParams
  - As usual for a Layout class ConstraintLayout contains a nested class ConstraintLayout.LayoutParams descended fromViewGroup.LayoutParams
    - This class contains mostly fields (instance variables) whose values allow a View to specify its size, position and behaviour wrt the containing ConstraintLayout
- Going forward
  - From the doco and the effort Google has spent updating the Layout Editor to support ConstraintLayout it can be deduced it’s intended to replace many of the existing Layout types
    - It does away with the need for deeply nested Layouts for a start but can also creates very fluid layouts

ConstraintLayout

- Basically like a RelativeLayout but
  - Views are attached to the layout sides or horizontal and vertical guidelines (like virtual layout sides) and other Views by software analogues of springs
  - These springs can expand and collapse depending of the viewport of the device they are being displayed on (including its current orientation)
  - The tension on springs holding a View between two endpoints can be biased towards one end by a percentage
  - Hard margins can be specified at the end point of each spring
  - There a lot more …

- Calculator Apps

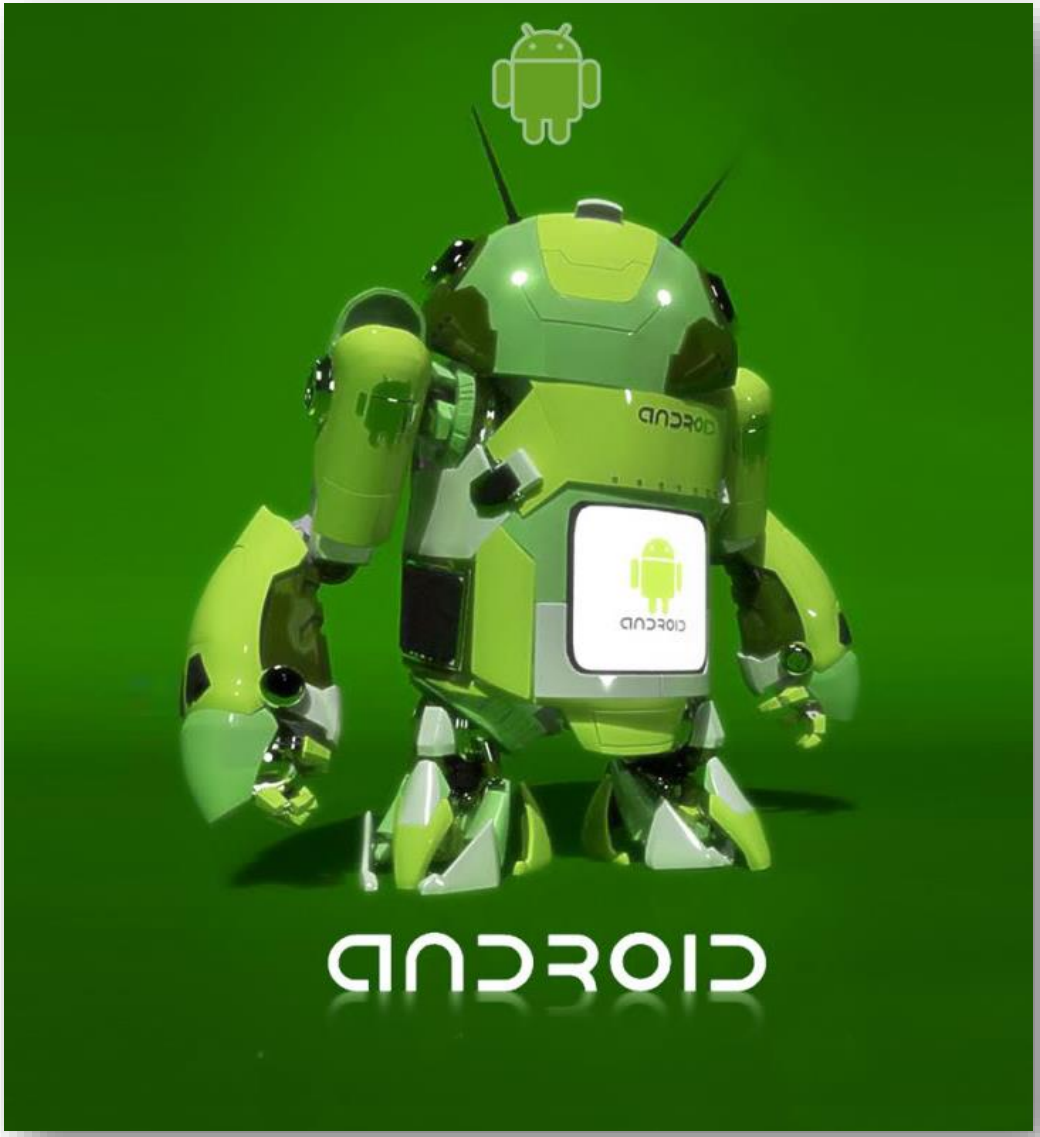


What You Should Understand

USE ANDROID DOCO IF YOU DON'T UNDERSTAND

- Layout
  - Everything
  - If you do not know the meaning of an attribute (layout_ or otherwise look it up in the doco)
    - e.g. android:stretchColumns
  - Just a reminder professional developers do not like using onClick
    - They prefer to create a listener containing an event handler and registering the listener to whatever objects is clicked to separate presentation and logic
    - No need for you to do this at the moment
- Activity Class
  - Everything
  - Including
    - The algorithm being used
    - The DecimalFormat class
    - Double.NaN
    - A TextView's setText and getText methods require and return a CharSequence respectively (it's an interface type)
      - It's an indirect subclass of String so you can supply a String to setText
      - If you need the result returned by getText to do something "Stringy" use the toString on the CharSequence value returned first

- Styles
- Themes
- Material Design



Styles

- Definition

Source: https://developer.android.com/guide/topics/ui/themes.html

  - "A style is a collection of attributes that specify the look and format for a View or window. A style can specify attributes such as height, padding, font color, font size, background color, and much more. A style is defined in an XML resource that is separate from the XML that specifies the layout.

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="@style/CodeFont"
    android:text="@string/hello" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Styles

Source: https://developer.android.com/guide/topics/ui/themes.html

- Style Inheritance
  - "The parent attribute in the <style> element lets you specify a style from which your style should inherit attributes. You can use this to inherit attributes from an existing style and define only the attributes that you want to change or add. You can inherit from styles that you've created yourself or from styles that are built into the platform."
  - So style cascades (as in CSS) are possible
- Why?
  - "Styles in Android share a similar philosophy to cascading stylesheets in web design—they allow you to separate the design from the content."

Themes

- Definition

Source: https://developer.android.com/guide/topics/ui/themes.html

  - "A theme is a style applied to an entire Activity or app [in the app's Manifest file], rather than an individual View, as in the example above. When a style is applied as a theme, every view in the activity or app applies each style attribute that it supports. For example, if you apply the same CodeFont style as a theme for an activity, then all text inside that activity appears in a green monospace font."
  - "Styles are cool. They allow you to define a set of attributes in one place and then apply them to as many widgets as you want. The downside of styles is that you have to apply them to each and every widget, one at a time. What if you had a more complex app with lots of buttons in lots of layouts? Adding your […] style to them all could be a huge task.
  - That is where themes come in. Themes take styles a step further: they allow you to define a set of attributes in one place, like a style – but then those attributes are automatically applied throughout your app.
  - Theme attributes can store a reference to concrete resources, such as colors, and they can also store a reference to styles. In a theme, you can say, for example, "I want all buttons to use this style." And you do not then need to find every button widget and tell it to use the theme."

Source: Phillips, Bill; Stewart, Chris. Android Programming: The Big Nerd Ranch Guide (p. 357). Pearson Education. Kindle Edition.

Material Design

* You can add many material design features to your app while maintaining compatibility with versions of Android earlier than 5.0. For more information, see Maintaining Compatibility. https://developer.android.com/training/material/get-started.html

- Definition
  - "Material design is a comprehensive guide for visual, motion, and interaction design across platforms and devices."
- Android and Material Design
  - "Android now includes support for material design apps. To use material design in your Android apps, follow the guidelines defined in the material design specification and use the new components and functionality available in Android 5.0 [Lollipop] (API level 21) and above*.
  - Android provides the following elements for you to build material design apps:
    - A new theme
    - New widgets for complex views [we are not talking calendar pickers!]
    - New APIs for custom shadows and animations
  - For more information about implementing material design on Android, see Creating Apps with Material Design."

See: https://developer.android.com/design/material/index.html

Material Design

- There are compatibility complications
  - e.g. you cannot use any Material theme with the standard Activity super class used for maximum backward compatibility, AppCompatActivity
    - The Material themes only work with Activities that are subclass of Activity which of course has limited backward compatibility
- Use v7 Support Libraries for [good but not complete] Material Design with backward compatibility to 2.3 (API 9)
  - "The v7 Support Libraries r21 and above includes the following material design features:
    - Material design styles for some system widgets when you apply one of the Theme.AppCompat themes.
    - Color palette theme attributes in the Theme.AppCompat themes.
    - The RecyclerView widget to display data collections.
    - The CardView widget to create cards.
    - The Palette class to extract prominent colors from images.
    - Updated support library versions may provide additional support for material design features. See the Support Library revision history for more information."

Source: https://developer.android.com/training/material/compatibility.html

Getting Most of MD with Compatibility to 2.3 (API 9)

build.gradle (Module: app)

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId "edu.monash.fab2081"
        minSdkVersion 19
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.2.0'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.android.support:design:25.2.0'
    testCompile 'junit:junit:4.12'
}
```

someActivity.java

```
public class MainActivity extends AppCompatActivity {
```

For backward compatibility including some MD support

This library includes many MD related classes but others need to be added for specialised widgets such as CardView and RecyclerView.

Getting Most of MD with Compatibility to 2.3 (API 9)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="edu.monash.fab2081">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Fab2081"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="Fab2081"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Defined by developer in styles.xml

Defined by developer in styles.xml

AndroidManifest.xml

The app bar, formerly known as the action bar in Android, is a special kind of toolbar that's used for branding, navigation, search, and actions.

The appcompat theme includes, for backward compatibility, a built-in Action Bar as part of the décor i.e. it's not a widget.

Since Lollipop the default theme has not included an Action Bar instead the developer should use a theme that supresses the Action Bar and use a widget called a Toolbar.
A particular Toolbar can be set to be an app's "Action Bar/App Bar using the setSupportActionBar method. When this is done

Getting Most of MD with Compatibility to 2.3 (API 9)

styles.xml

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>

    <style name="AppTheme.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar" />

    <style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

</resources>
```

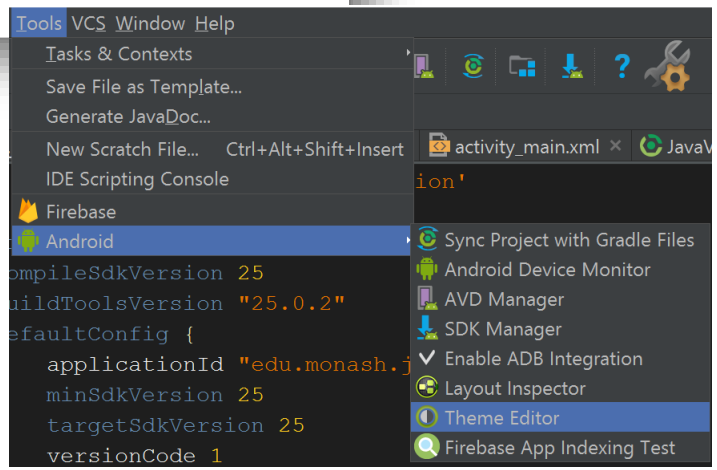
Defining AppTheme by inheriting an appcompat theme and overriding (redefining) the 3 most significant colour styles in that theme with custom colours

Creating a theme that supresses the pre-lollipop Action Bar decor

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Defining 3 custom colour values

colors.xml



Tools → Android → Theme Editor

Deeper into Material Design

NOT PRESCRIBED
PRE_READING

- Links
  - Overview

    See: https://developer.android.com/design/material/index.html
  - Getting Started

    See: https://developer.android.com/training/material/get-started.html
  - Training Overview page with many links

    See: https://developer.android.com/training/material/index.html
  - Material Design Theme – What? Customising it.

    See: https://developer.android.com/training/material/theme.html
  - Compatibility – Full MD >= API 21, Most of MD < API 21

    See: https://developer.android.com/training/material/compatibility.html

Action Bar vs AppBar

Source: https://material.io/guidelines/layout/structure.html#structure-toolbars

- This whole area is an historical mess.
  - "The app bar, formerly known as the action bar in Android, is a special kind of toolbar that's used for branding, navigation, search, and actions."
  - App Bar and Action Bar tend to be synonyms in the Android community.
- There is no UI component called App Bar or Action Bar. It's a UI design concept. Definition and key features are here
- Pre-Lollipop
  - Action Bar was part of the default theme, so not a widget
    - Probably to force consistency on this core UI feature
- Post-Lollipop
  - Action Bar not part of default theme
  - Developers expected to work with a designated Toolbar widget instead
    - Nearly always enclosed in an AppBarLayout which is a vertical LinearLayout which implements many of the features of material design's app bar concept
    - "In this release [lollipop], Android introduces a new Toolbar widget. This is a generalization of the Action Bar pattern that gives you much more control and flexibility. Toolbar is a view in your hierarchy just like any other, making it easier to interleave with the rest of your views, animate it, and react to scroll events. You can also set it as your Activity's action bar, meaning that your standard options menu actions will be display[ed] within it."

Source: here

Action Bar vs AppBar

- There is no UI component called App Bar or Action Bar
  - There is a UI components called Toolbar and confusingly a class called ActionBar which addresses whatever is the App Bar/Action Bar for an Activity's UI (could be implemented as an AppBar or Toolbar).
  - android.support.v7.widget.Toolbar class
    - "A Toolbar is a generalization of action bars for use within application layouts. While an action bar is traditionally part of an Activity's opaque window decor controlled by the framework, a Toolbar may be placed at any arbitrary level of nesting within a view hierarchy. An application may choose to designate a Toolbar as the action bar for an Activity using the setSupportActionBar() method." Source: here
  - setSupportActionBar(Toolbar toolbar) Method
    - "When set to a non-null value the getActionBar() method will return an ActionBar object that can be used to control the given toolbar as if it were a traditional window decor action bar." Source: here
    - In addition to the Toolbar's methods

Action Bar vs AppBar

- Using an ActionBar as your App Bar (not preferred)
  - Use a theme which does NOT contain ".NoActionBar"
  - The Action bar will appear without any coding but use getSupportActionBar() to gain a reference to it so it can be manipulated
  - There are API level differences as features were incrementally added
- Using a Toolbar as your App Bar (preferred)
  - Use a theme which DOES contain ".NoActionBar" so there is no ActionBar
    - Or using style item elements in a style's XML definition
  - Use an android.support.v7.widget.Toolbar instead and use setSupportActionBar (Toolbar toolbar) to set the Toolbar to act as the Action Bar for this Activity window.
  - There are no API differences
- Setting up a Toolbar based App Bar
  - See: https://developer.android.com/training/appbar/index.html
- Some relevant videos
  - here and here

Warning:

these videos contain extremely annoying but knowledgeable people