http://www.android.com/

UI (User Interface) overview
Supporting Multiple Screens
Touch events and listeners

User Interface Layout

- The Android user interface (UI) consists of screen views (one or more viewgroups of widgets), screen touch events, and key presses
- Resource directories – the /app/res/* folders in the project view
- General UI attributes - Each UI object has 3 definable attributes that customize the look and feel of the UI
  - The dimension of the object: **dp** (density independent pixels), px, in, mm, pt and **sp** (scale independent pixels – as dp but scaled to font preference)
  - Text in the object (supports formatted strings, html and special chars)
  - The color of the object
- Alternate resources/localization (use the two first letters in language)
  - res/values-sv/strings.xml, res/values-da/strings.xml etc.
  - Missing values will fall back to the default res/values/strings.xml
- Multiple screen pixel densities (pixel count/screen size) in dpi (dots per inch)
  - res/drawable-x: ldpi(120), mdpi(160), hdpi(240), xhdpi(320), xxhdpi(480) and xxxhdpi(640)
  - Android at run-time determines the closest one to use and scales them
  - res/drawable is used for bitmaps which should not scale

Layouts and attributes

- The user interface for Activities is defined via layouts. The layout defines the UI elements, their properties and their arrangement
  - A layout can be defined by XML, via Java code or a mix of both
  - The XML way is preferred since it is more flexible and easy to extend, customize and change - compared to using Java code during run-time
- These definitions are placed in the <projname>/res/layout/main.xml file and connected to the user interface if needed
  - Usually there is one /res/layout/<file>.xml file for every Activity screen
  - Symbol @ means that string should be expanded by the XML parser
  - Symbol + means that a unique ”id” should be created in the name space

Resource	Reference in Java	Reference in XML
res/layout/main.xml	`R.layout.main`	`@layout/main`
res/drawable-hdpi/icon.png	`R.drawable.icon`	`@drawable/icon`
`@+id/home_button`	`R.id.home_button`	`@id/home_button`
`<string name="hello">`	`R.string.hello`	`@string/hello`

Alternative resources and localization

http://developer.android.com/guide/topics/resources/index.html

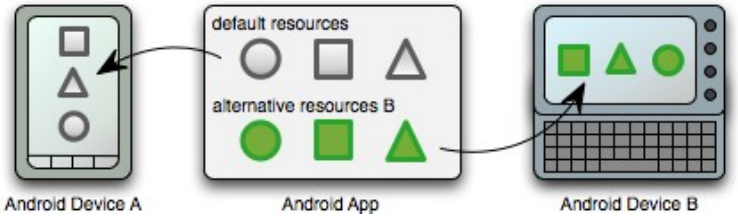
- Android run on many devivces and in many regions. To reach the most users, your application should handle text, audio files, numbers, currency, and graphics in ways appropriate to the locales and screen size etc.
- The default resources are required and should define every string etc.

```
The text strings in res/values/strings.xml should use the default language, which is the language that you
expect most of your application's users to speak.
res/values/strings.xml (required directory)
The default resource set must also include any default drawables and layouts, and can include other types
of resources such as animations.
res/drawable/(required directory holding at least one graphic file?)
res/layout/ (required directory holding an XML file that defines the default layout)
res/anim/ (required if you have any res/anim-<qualifiers> folders)
res/xml/ (required if you have any res/xml-<qualifiers> folders)
res/raw/ (required if you have any res/raw-<qualifiers> folders)
```

- When the user runs your program the Android system selects which resources to load, based on the device's locale etc.
  - If not found or partially not found it will fallback to the default resources
- Examples for lang, dimensions, orientation and styles etc.

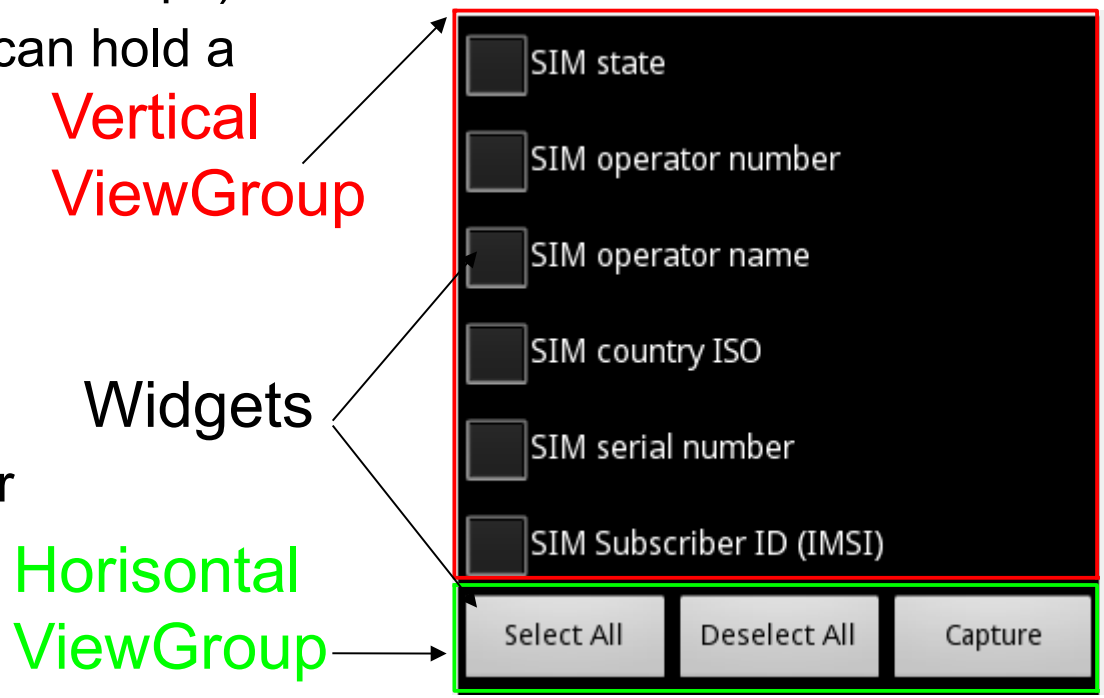
res/layout-se/main.xml, res/values-se/strings.xml

res/values-ldpi/dimens.xml, res/values-9/styles.xml



Views and ViewGroups

- The basic building block of a graphical layout is a View
- Each **View** is described by a View Object, which is responsible for drawing a rectangular area and **handling events** in that area
- The View is a base class for objects that interact with the user; they are called widgets
- A ViewGroup Object is a type of View that acts as a container to hold multiple Views (or other ViewGroups)
- For example a ViewGroup can hold a vertical or horizontal placement of views and widgets, as shown in the figure
- The ViewGroup is a base class for screen layouts
- The **layout** defines the user interface elements, their positions, and their actions

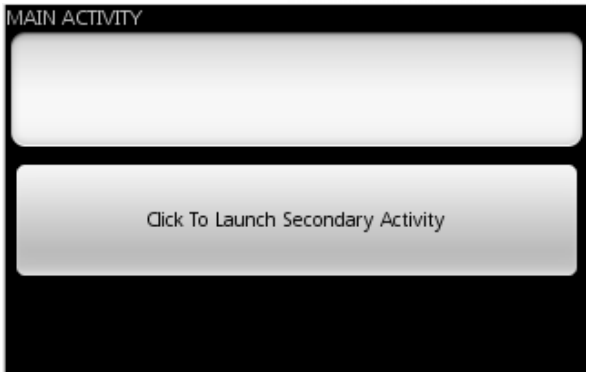


Layout and Views 1

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout android:id="@+id/TableLayout01"
	android:layout_width="fill_parent" android:layout_height="fill_parent"
	xmlns:android="http://schemas.android.com/apk/res/android">
	<TableRow android:id="@+id/TableRow01" android:layout_width="wrap_content"
		android:layout_height="wrap_content">
		<Button android:text="@+id/Button01" android:id="@+id/Button01"
			android:layout_width="wrap_content"
			android:layout_height="wrap_content" />
		<CheckBox android:text="@+id/CheckBox01" android:id="@+id/CheckBox01"
			android:layout_width="wrap_content"
			android:layout_height="wrap_content" />
	</TableRow>
	<TableRow android:id="@+id/TableRow02"
		android:layout_width="wrap_content"
		android:layout_height="wrap_content">
		<TextView android:text="@+id/TextView01"
			android:id="@+id/TextView01"
			android:layout_width="wrap_content"
			android:layout_height="wrap_content" />
		<TextView android:text="@+id/TextView02"
			android:id="@+id/TextView02"
			android:layout_width="wrap_content"
			android:layout_height="wrap_content" />
	</TableRow>
	<TableRow android:id="@+id/TableRow03"
		android:layout_width="wrap_content"
		android:layout_height="wrap_content">
		<TimePicker android:id="@+id/TimePicker01"
			android:layout_width="wrap_content"
			android:layout_height="wrap_content" />
	</TableRow>
	<Spinner android:id="@+id/Spinner01"
		android:layout_width="wrap_content"
		android:layout_height="wrap_content" />
	<VideoView android:id="@+id/VideoView01"
		android:layout_width="wrap_content"
		android:layout_height="wrap_content" />
</TableLayout>
```

Layouts, resources and code

- The res/values/strings.xml normally contains all predefined strings in the project
- Activitys onCreate(), strings.xml and main.xml



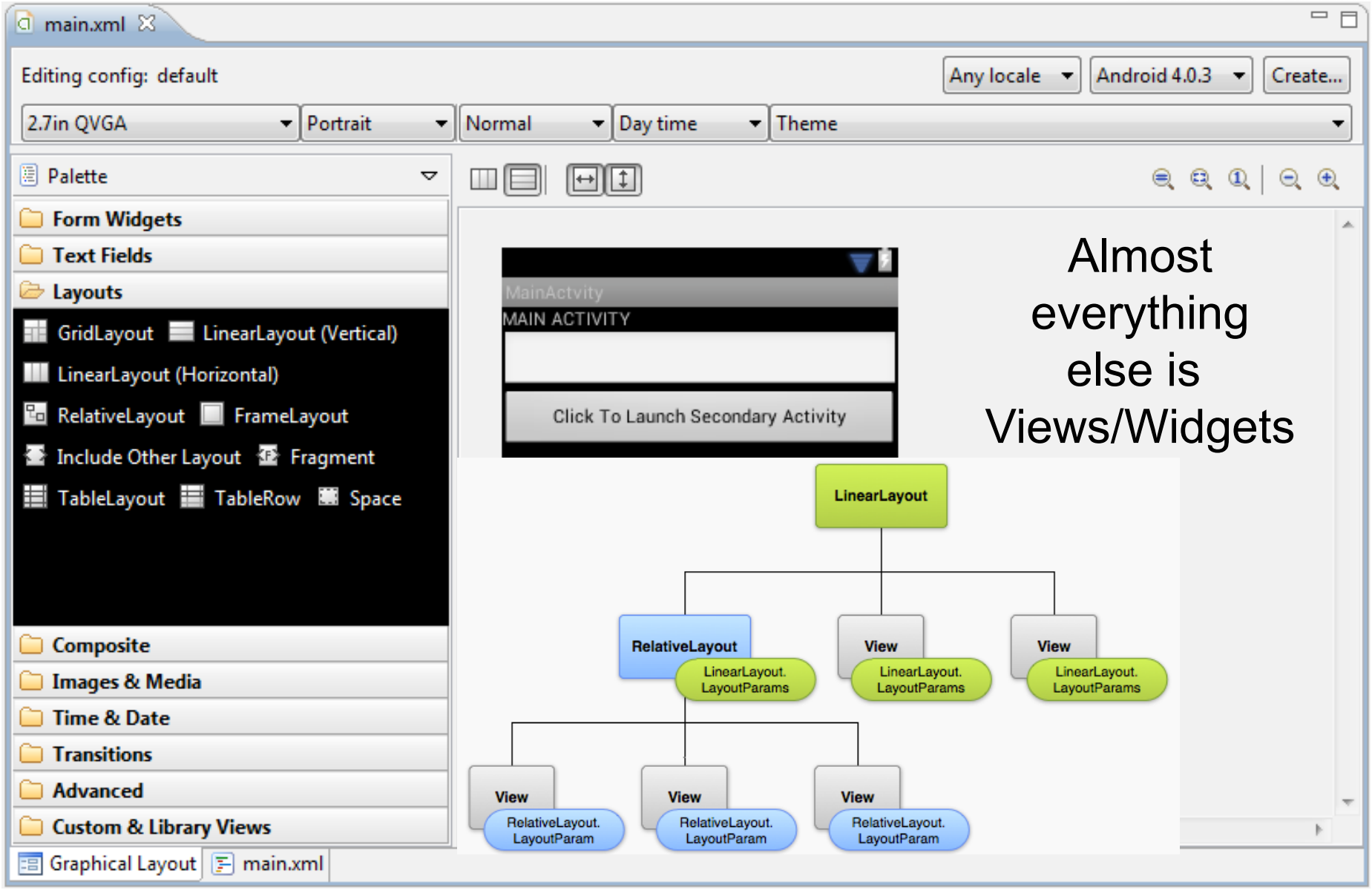

```
public class MainActivity extends Activity
implements View.OnClickListener
{

   private EditText mEditText1;
   private Button mButton1;

/** Called when the activity is first
created. */

@Override
public void onCreate(Bundle
savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   mEditText1 =
(EditText)findViewById(R.id.editText1);

   mButton1 =
(Button)findViewById(R.id.button1);

   mButton1.setOnClickListener(this);
}
//...
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <string name="hello">Hello World, MainActvity!</string>
      <string name="app_name">MainActvity</string>
      <string name="mainactivity">MAIN ACTIVITY</string>
<string name="btnLaunchSecAct">Click To Launch Secondary Activity</string>
</resources>
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/mainactivity"/>
    <EditText android:id="@+id/editText1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    <Button android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/btnLaunchSecAct"/>
</LinearLayout>
```

main.xml

Layouts and Views 2



Layouts and Views examples

Layouts

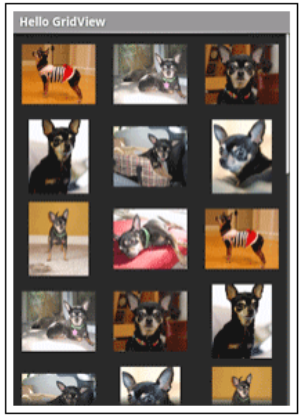
In general a view uses an Adapter to bind data to its layout

Linear Layout

Relative Layout

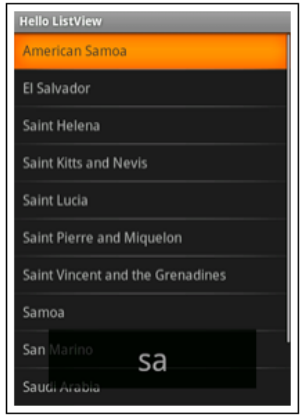
Table Layout

Grid View



Tab Layout

List View



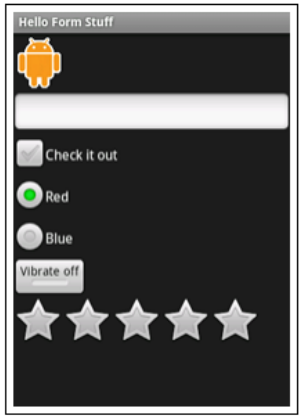
http://developer.android.com/resources/tutorials/views/index.html

Widgets & Other Views

Date Picker

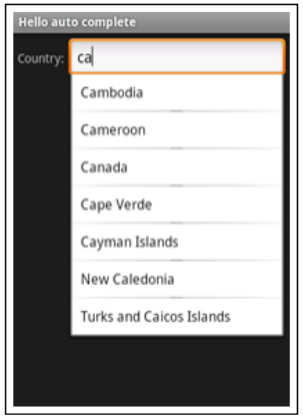
Time Picker

Form Stuff



Spinner

Auto Complete



Gallery

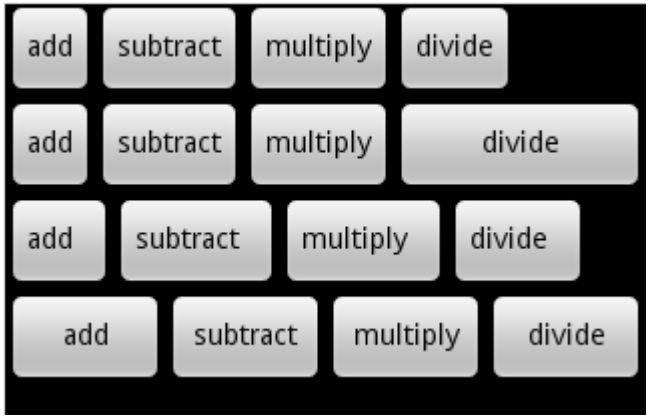
Controlling the Width and Height of UI Elements

- Each View object must specify a total width **android:layout_width** and total height **android:layout_height** in one of three ways
  - **exact dimension** - Provides control, but does not scale to multiple screen types well
  - **wrap_content** - Just big enough to enclose the contents of the element plus padding
  - **match_parent** (named **fill_parent** before API 8) - Size maximized to fill the element's parent, minus padding
- **Margins** (inside of an UI element border) is part of the size and **Padding** (outside of the UI element border) is specified as dp, it can be specified using one of two types of attributes
  - All - Sets margins/padding equal on all four sides of an element
  - Left, Right, Top, Bottom - Sets margin/padding on each side of an element separately
- Another attribute is **android:layout_weight**, which can be assigned a number. It provides the Android system with a way to determine relative importance between different elements of a layout
  - **Note!** layout_width or layout_height should be set to ”0” in this case

Controlling W & H cont.

In the fourth row/tag below (Layout5), all buttons use layout_width=”0dp”, also add **layout_weight** and assign it the same value for all buttons

This gives the most satisfying layout!



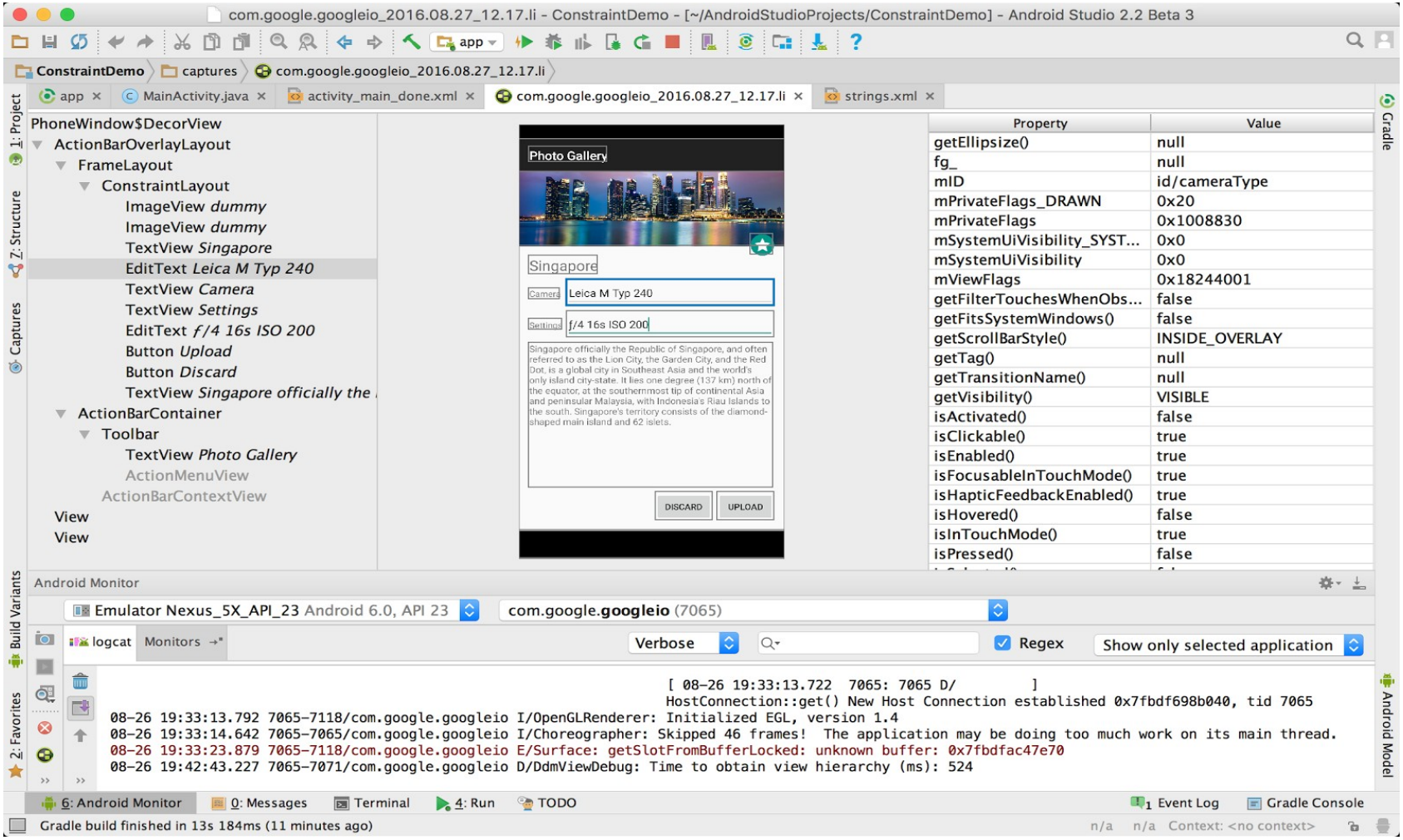

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout1" android:orientation="vertical"
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent" android:layout_height="fill_parent">
      <LinearLayout android:id="@+id/linearLayout2"
            android:layout_width="fill_parent" android:layout_height="wrap_content">
            <Button android:text="add" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" />
            <Button android:text="subtract" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" />
            <Button android:text="multiply" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" />
            <Button android:text="divide" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" />
      </LinearLayout>
      <LinearLayout android:id="@+id/linearLayout3"
            android:layout_width="fill_parent" android:layout_height="wrap_content">
            <Button android:text="add" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" />
            <Button android:text="subtract" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" />
            <Button android:text="multiply" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" />
            <Button android:text="divide" android:layout_width="fill_parent"
                  android:layout_height="wrap_content" />
      </LinearLayout>
      <LinearLayout android:id="@+id/linearLayout4"
            android:layout_width="fill_parent" android:layout_height="wrap_content">
            <Button android:text="add" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:paddingRight="20sp"/>
            <Button android:text="subtract" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:paddingRight="20sp"/>
            <Button android:text="multiply" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:paddingRight="20sp"/>
            <Button android:text="divide" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:paddingRight="20sp"/>
      </LinearLayout>
      <LinearLayout android:id="@+id/linearLayout5"
            android:layout_width="fill_parent" android:layout_height="wrap_content">
            <Button android:text="add" android:layout_width="0dp"
                  android:layout_height="wrap_content" android:layout_weight="1" />
            <Button android:text="subtract" android:layout_width="0dp"
                  android:layout_height="wrap_content" android:layout_weight="1" />
            <Button android:text="multiply" android:layout_width="0dp"
                  android:layout_height="wrap_content" android:layout_weight="1" />
            <Button android:text="divide" android:layout_width="0dp"
                  android:layout_height="wrap_content" android:layout_weight="1" />
      </LinearLayout>
</LinearLayout>
```

Layout Inspector (Hierarchy Viewer)

View the layout in a program. Only works with the current app in emulator.

Above AS 2.2 - Tools > Android > 

http://tools.android.com/tech-docs/layout-inspector



Set Relative Layout and Layout ID

Sometimes it is more convenient to set the layout relative to a starting object or parent object rather than absolute rules

Also, if the UI starts nesting LinearLayouts, it might be simpler to use relative layouts. This can be done using a RelativeLayout view


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="fill_parent" android:layout_height="wrap_content" android:padding="10px">

       <TextView android:id="@+id/label" android:layout_width="fill_parent"
              android:layout_height="wrap_content" android:text="Type here:" />

       <EditText android:id="@+id/entry" android:layout_width="fill_parent"
              android:layout_height="wrap_content" android:background="@android:drawable/editbox_background"
              android:layout_below="@id/label" />

       <Button android:id="@+id/ok" android:layout_width="wrap_content"
              android:layout_height="wrap_content" android:layout_marginLeft="10px" android:text="OK"
              android:layout_alignParentRight="true" android:layout_below="@id/entry" />

       <Button android:layout_width="wrap_content"
              android:layout_height="wrap_content" android:text="Cancel"
              android:layout_toLeftOf="@id/ok" android:layout_alignTop="@id/ok"  />
</RelativeLayout>
```

Should be dp and sp for text size!

Some rules for children in a Relative Layout

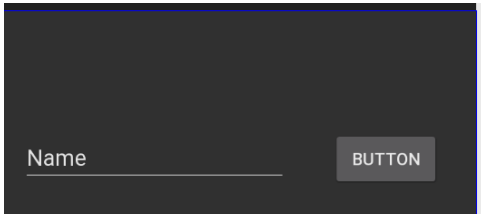
```
android:layout_below="@id/TextView1"
```

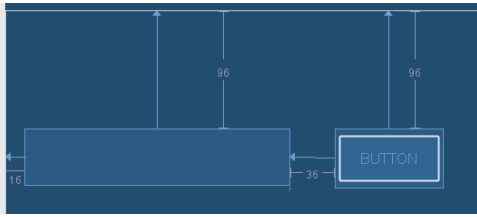
Relative Layout Rule	XML Attribute (All Start with the `android:` Tag)	Java Constant
Align this view's edge relative to anchor view's edge	`layout_above`	`ABOVE`
	`layout_below`	`BELOW`
	`layout_toRightOf`	`RIGHT_OF`
	`layout_toLeftOf`	`LEFT_OF`
Align this view's edge with anchor view's edge	`layout_alignTop`	`ALIGN_TOP`
	`layout_alignBottom`	`ALIGN_BOTTOM`
	`layout_alignRight`	`ALIGN_RIGHT`
	`layout_alignLeft`	`ALIGN_LEFT`
Align this view's text baseline with anchor view's text baseline	`layout_alignBaseline`	`ALIGN_BASELINE`
Align this view's edge with parent view's edge	`layout_alignParentTop`	`ALIGN_PARENT_TOP`
	`layout_alignParentBottom`	`ALIGN_PARENT_BOTTOM`
	`layout_alignParentRight`	`ALIGN_PARENT_RIGHT`
	`layout_alignParentLeft`	`ALIGN_PARENT_LEFT`
Center this view within parent	`layout_centerInParent`	`CENTER_IN_PARENT`
	`layout_centerHorizontal`	`CENTER_HORIZONTAL`
	`layout_centerVertical`	`CENTER_VERTICAL`

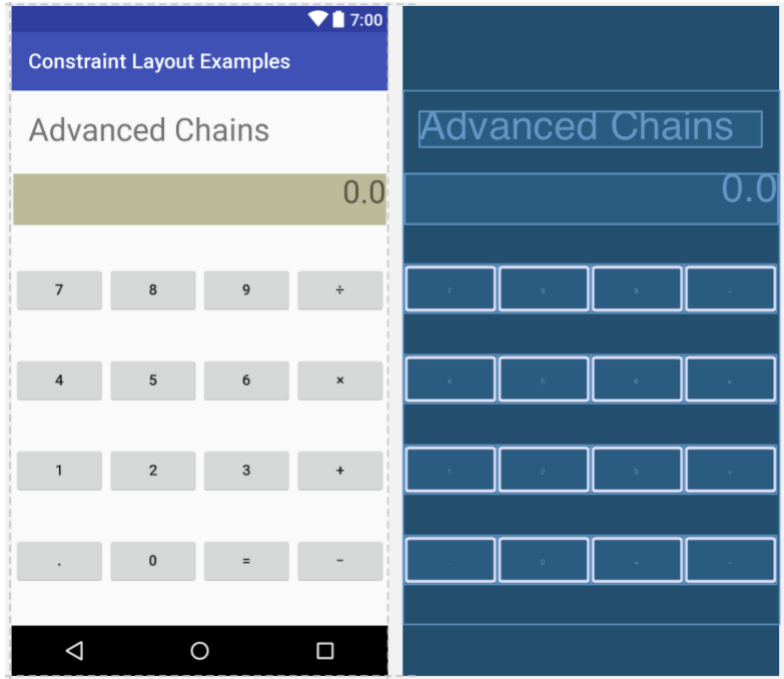
Constraint Layout

- Allows you to create large and complex layouts with a flat view hierarchy by adding vertical and horizontal constraints. Loads faster than traditional layouts
- It is similar to RelativeLayout but more flexible and easier to use. With chains (spread, spread inside, weighted and packed) advanced views are possible
- Training/examples: https://developer.android.com/training/constraint-layout/

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="36dp"
        android:layout_marginTop="96dp"
        android:text="Button"
        app:layout_constraintStart_toEndOf="@+id/editText"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="96dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="Name"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```







Useful UI (TextView) Attributes

```
<TextView android:text="@string/myTextString" android:id="@+id/my_text_label"
      android:background="@android:drawable/editbox_background" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:textSize="48sp" />
```

Dimension

Possible Values

Any number followed by one of the following dimensions:

px—Actual pixels on the screen

dp (or **dip**)—Device-independent pixels relative to a 160dpi screen

sp—Device-independent pixels scaled by user's font size preference

in—Inches based on physical screen size

mm—Millimeters based on physical screen size

pt—1/72 of an inch based on physical screen size

Color

Possible values are a 12-bit color #rgb, 16-bit color with alpha opacity #argb, 24-bit color #rrggbb, or 32-bit color with alpha opacity #aarrggbb. It is also possible to utilize the predefined colors in the Color class within Java files, such as Color.CYAN, Color.GREEN, etc.

Default Values in Bold in the Last Column

TextView Attribute	XML Element	Java Method	Possible and Default Values
Display string	android:text	setText(CharSequence)	Any string
Font size	android:textSize	setTextSize(float)	Any dimension
Font color	android:textColor	setTextColor(int)	Any color
Background color	N/A	setBackgroundColor(int)	Any color
Font style	android:textStyle	setTypeface(Typeface)	bold italic bold italic
Font type	android:typeface	setTypeface(Typeface)	**normal** sans serif monospace
Text placement in display area	android:gravity	setGravity(int)	top bottom **left** right (more...)

Layouts and Views tips

http://developer.android.com/guide/topics/manifest/activity-element.html

- To force single instance or single task mode - set the following for the Activity in the MAIN/LAUNCHER intent-filter in androidmanifest.xml

```
android:launchMode="singleInstance"
android:launchMode="singleTask"
android:launchMode="singleTop"
```

The "singleInstance" mode is identical to "singleTask" except if "singleInstance" starts another activity, that activity will be launched into a different task regardless of its launch mode.

- Retain the task state - return to the last state at re-launch

```
android:alwaysRetainTaskState="true"
```

- To force screen orientation - add to the Activity element

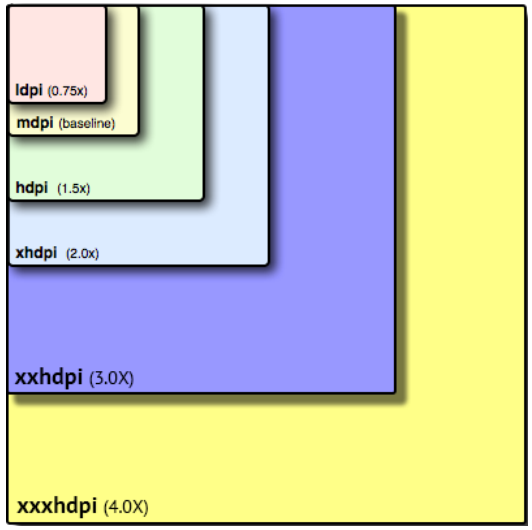
```
android:screenOrientation="portrait"
android:screenOrientation="landscape"
```

- Try out the Api Demos app in the emulator to examine all possible GUI combinations/controls - source at:
  - https://android.googlesource.com/platform/development/ > version > samples
- User Interface Guidelines for the interaction and visual design of Android applications
  - New: http://developer.android.com/design/index.html
- http://petrnohejl.github.io/Android-Cheatsheet-For-Graphic-Designers/

Supporting Multiple Screen Sizes 1

http://developer.android.com/guide/practices/screens_support.html

- Density-independent pixel (dp) - A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way (dpi/ppi = dots/pixels per inch)
  - For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels → pixels per inch = dp * (dpi / 160). For a xxhdpi (3.0x) screen it is 3 pixels
- Layout-small, layout-large or layout-xlarge is deprecated
- Use smallestWidth sw<N>dp after Android 3.2 – note **dp**
  - For example: res/layout-sw600dp/ - defines the smallest available width required by your layout resources



Baseline = mdpi = 160 dpi

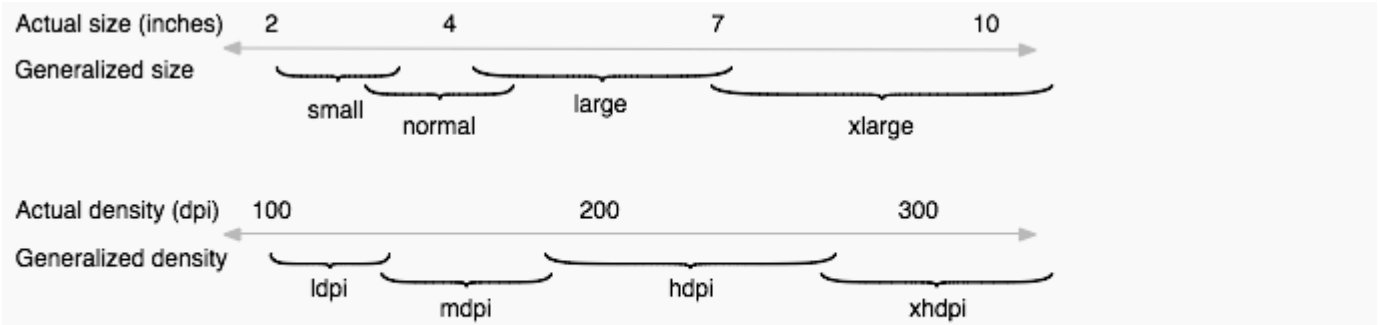


Figure 1. Illustration of how Android roughly maps actual sizes and densities to generalized sizes and densities (figures are not exact).

Supporting Multiple Screen Sizes 2

http://developer.android.com/guide/practices/screens_support.html

- We also have w<N>dp and h<N>dp which is minimum available width and height in dp units
- Example 800x1280px 7” tablet → 215,6 ppi falls into hdpi (1,5x)
  - We should use 800/1,5 = sw533dp, but if we set layout-sw480dp we handle 720x1280 pixel tablets as well
- 1080x1920px 5,2” phone → 423,6 ppi falls into xxhdpi (3.0x)
  - 1080/3 = 360 so if we would like another layout on this phone we must set layout-sw360dp
- We could also use the w<N>dp (1920/3 = 640), setting layout-w600dp gives any screen with 600dp available width or more the desired layout whether the device is in landscape or portrait orientation

Calculation of screen PPI (Pixels Per Inch)

1. Calculate diagonal resolution in pixels using the Pythagorean theorem:

$$d_p = \sqrt{w_p^2 + h_p^2}$$

2. Calculate PPI:

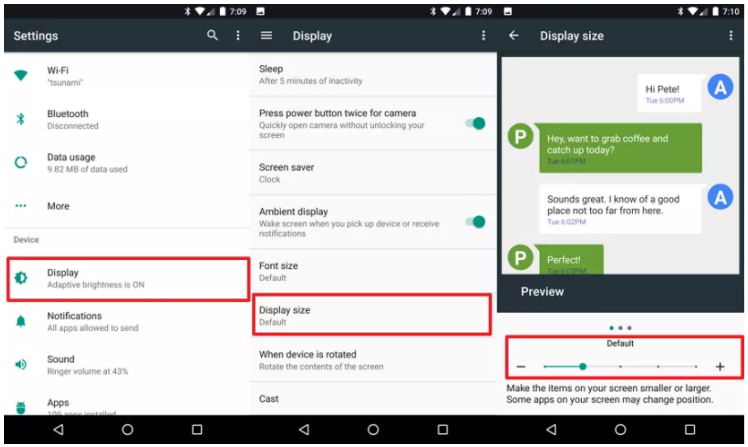
$$PPI = \frac{d_p}{d_i}$$

where

- d_p is diagonal resolution in pixels
- w_p is width resolution in pixels
- h_p is height resolution in pixels
- d_i is diagonal size in inches (this is the number advertised as the size of the display).

Supporting Multiple Screen Sizes Wrap-up

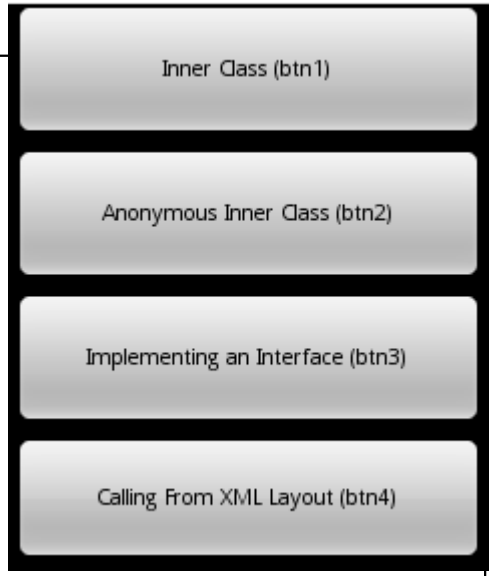
- Get and set the density
  - From a command shell get the density with: adb shell wm density
  - To set the density just put a number after as: adb shell wm density 320
  - With Android Nougat users can control the DPI under Settings > Display > Display size



- Calculate the density
  - The physical density of a 5.2 inch screen with 1920 x 1080 is 480
    - This translates into: 480/160 = xxhdpi 3
  - The formula is actual-dpi / 160. (Everything is scaled to 160 dpi)
  - To get the physical density = sqrt((wp * wp) + (hp * hp)) / di
  - Where:
    - wp is width resolution in pixels, hp is height resolution in pixels, and di is diagonal size in inches. Example a 8.3 inch tablet with 1920 x 1200 pixels
    - (1200^2 + 1920^2)^0.5 = 2265 / 8.3 = 273 dpi or ppi
    - 273/160 = 1.7. Will fall into xhdpi 2 or hdpi 1.5 (1200/2 = sw-600dp)

Handle View touch events 1

- There are four different ways to add listeners for handling touch events (button onClick and onLongClick)




```
@Override
public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);

     // method 1 - uses an inner class named btn1Listener...
     Button btn1 = (Button)findViewById(R.id.Button01);
     btn1.setOnClickListener(btn1Listener);

     // method 2 - use an anonymous inner class as a listener...
     Button btn2 = (Button)findViewById(R.id.Button02);
     btn2.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
               showToastMessage("You clicked btn2 - uses an anonymous inner class");
          }
     });

     // method 3 - note that this activity implements the View.OnClickListener interface
     // which means that we must implement the onClick() method (which you'll find below)..
     Button btn3 = (Button)findViewById(R.id.Button03);
     btn3.setOnClickListener(this);
     // method 4 - look at the method btn4Listener() below
}
```

Handle View touch events 2

- See the ButtonClickTest project

```
// Method 1 - here's the inner class used as a listener for btn1...
private View.OnClickListener btn1Listener = new View.OnClickListener() {
     @Override
     public void onClick(View v) {
          showToastMessage("You clicked btn1 - uses an inner class named btn1Listener");
     }
};

// Method 3 - here's a method that you must have when your activity implements the
// public class ButtonClickTest extends Activity implements View.OnClickListener interface...
@Override
public void onClick(View v) {
     showToastMessage("you clicked on a btn3, which uses this Activity as the listener");
}

// Method 4 - here's the handler for btn4 (declared in the XML layout file, btn4 properties)
// note: this method (On Click property) only works with android api level 7 and higher,
// it must be public and must take a single parameter which is a View
public void btn4Listener(View v) {
     showToastMessage("You clicked btn4 - listener was set up in the XML layout");
}

private void showToastMessage(String msg){
     Toast toast = Toast.makeText(this /*getApplicatonContext()*/, msg, Toast.LENGTH_SHORT);
     toast.show();
}
```

Handle View touch events 3

- Examples using the (View v) parameter

```
// working with just one clicklistener and if you have declared
// b*.setOnClickListener(myOnlyhandler);, you can do...
private View.OnClickListener myOnlyhandler = new View.OnClickListener() {
     public void onClick(View v) {
          if( button1.getId() == ((Button)v).getId() ){
               // it was the first button
          }
          else if( button2.getId() == ((Button)v).getId() ){
               // it was the second button
          }
     }
}
// if you defined android:onClick="ButtonOnClick" for all buttons in your XML, you can do...
public void ButtonOnClick(View v) {
     switch (v.getId()) {
        case R.id.button1:
          doSomething1();
          break;
        case R.id.button2:
          doSomething2();
          break;
     }
}
```

Event Listeners interfaces and methods

- onClick() from View.OnClickListener
- onLongClick() from View.OnLongClickListener
- onFocusChange() from View.OnFocusChangeListener
- onKey() from View.OnKeyListener
  - Called when the view has focus and a key is pressed
- onTouch() From View.OnTouchListener – gestures
- Methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction
- More about Views in the API documents
  - http://developer.android.com/reference/android/view/View.html

The Application Context

- The class android.content.Context provides the connections to the current ”context”
- As Activities and Services extends this class you can directly access the Context via
  - this (current Context)
  - getApplicationContext()
  - Or explicitly use ClassName.this

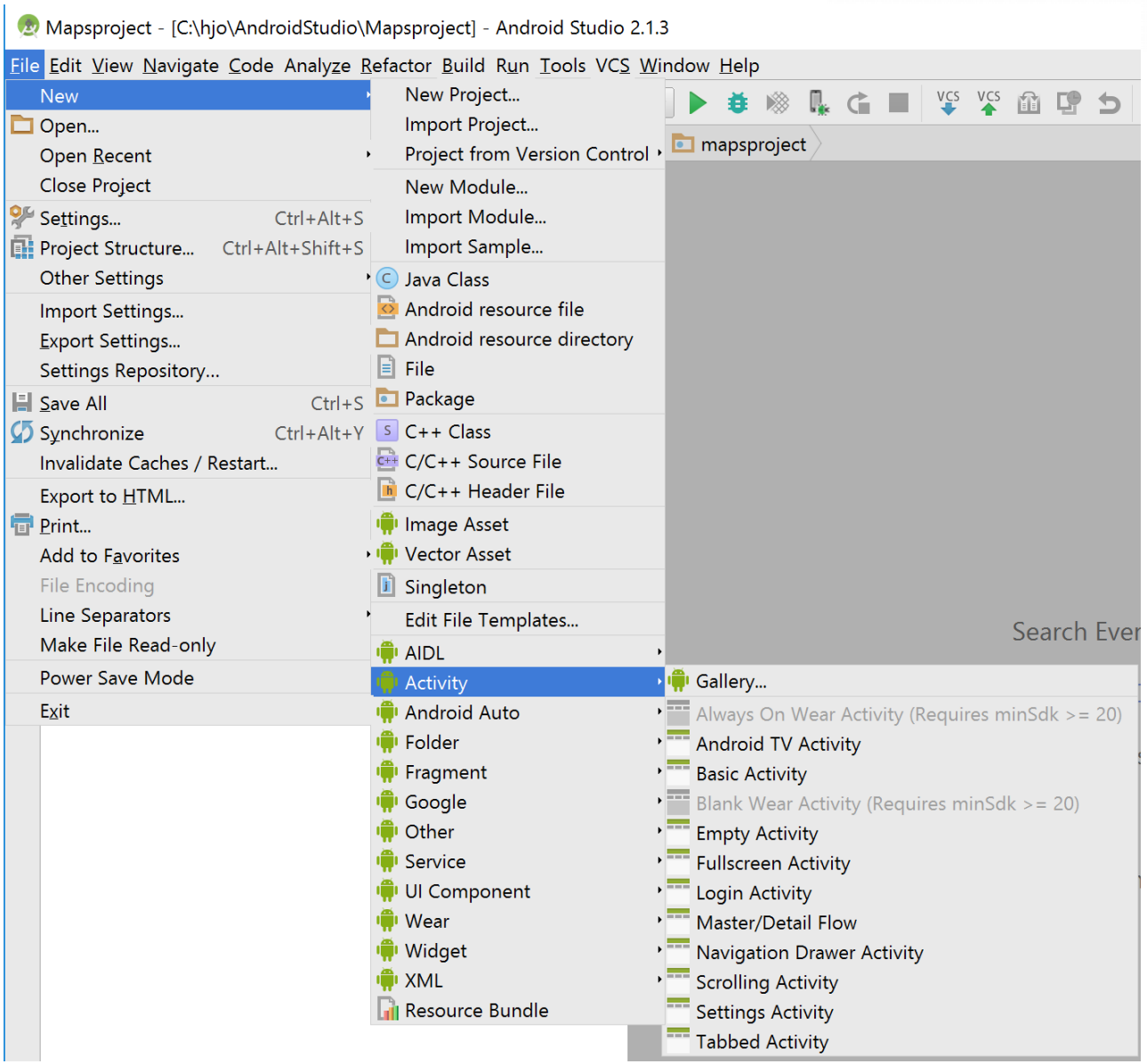
Return the context of the single, global Application object of the current process.
This generally should only be used if you need a Context whose lifecycle is separate from the current context, that is tied to the lifetime of the process rather than the current component.

- Using the this reference to current object is recommended but it is not always resolved correctly
  - For example in inner classes and anonymous inner classes as in the previous ButtonClickTest example
- The static abstract Context class is usually passed to the getSystemService() which allows to receive a manager object for different hardware parts in the system as Sensors etc.

```
String ns = Context.NOTIFICATION_SERVICE;
mNManager = (NotificationManager) getSystemService(ns);
```

AS File > New > ...

- Or right click on an object any-where in the AS environment (folders, files, UI, etc.)
- Wizards for most common stuff
  - Layouts, XML values file, Service, ContentProvider, …



Eclipse File > New > Other...

- Wizards for most common stuff
  - Layouts, XML values file, Service, ContentProvider, ...