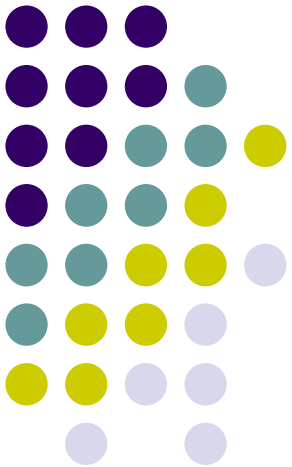
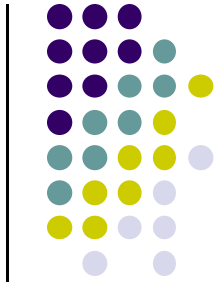
CS 528 Mobile and Ubiquitous Computing

Lecture 2: Intro to Android Programming

Emmanuel Agu

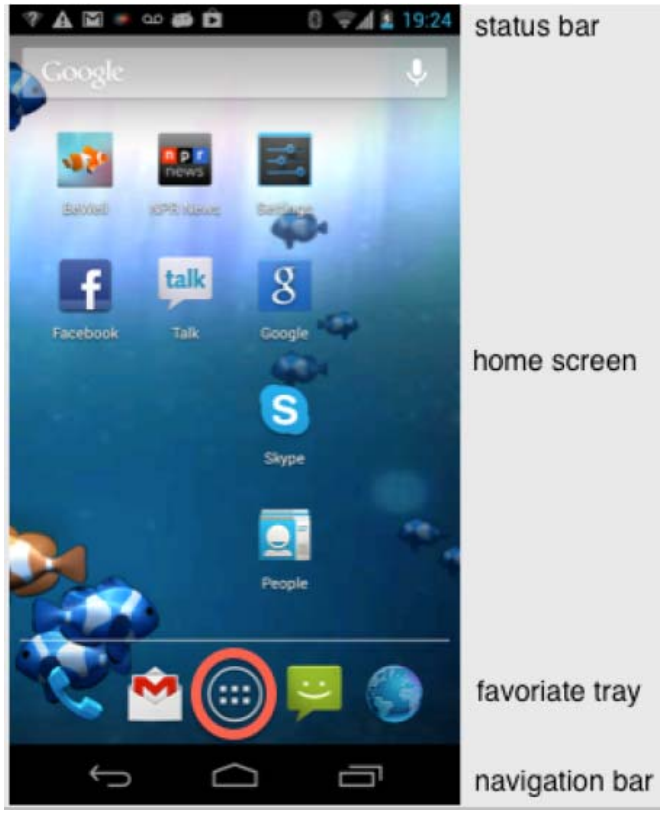




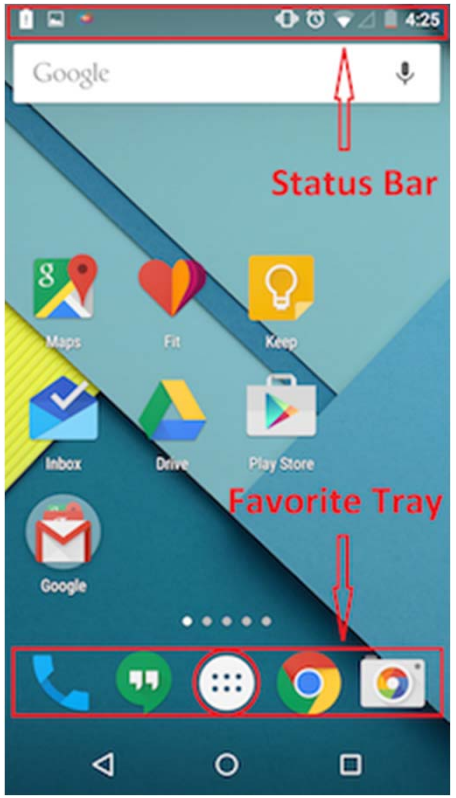
Android UI Tour

Home Screen

- First screen after unlocking phone or hitting **home** button
- Includes **favorites** tray (e.g phone, mail, messaging, web, etc)



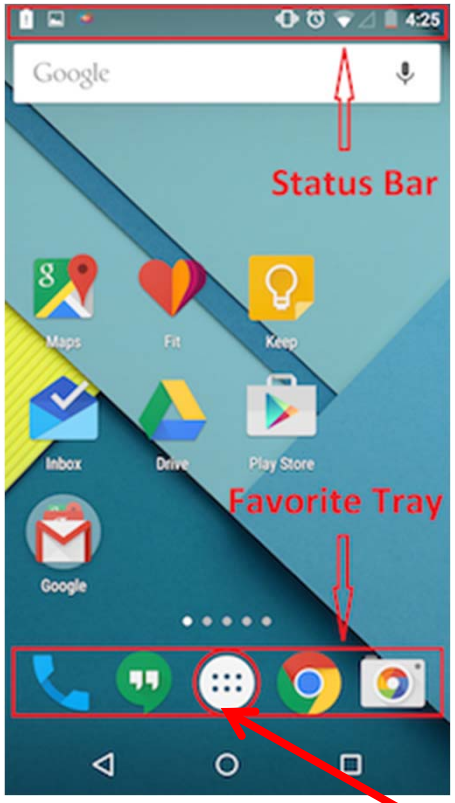
Android 4.0



Android 5.0

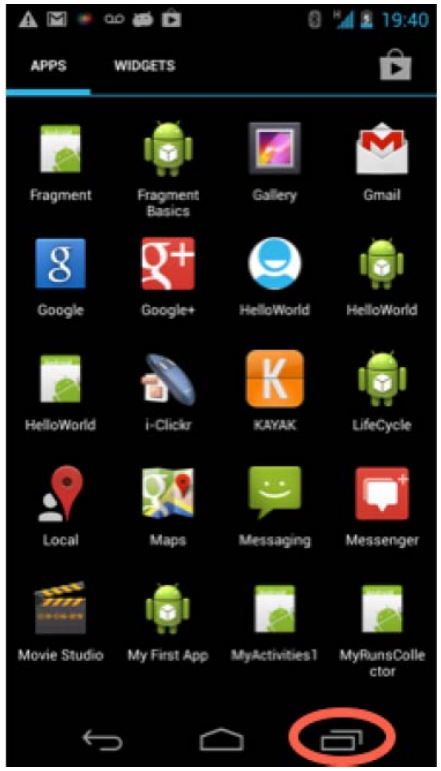
All Apps Screen

- Accessed by touching **all apps button** in favorites tray
- Users can swipe through multiple app and widget screens
- Can customize by dragging and dropping items

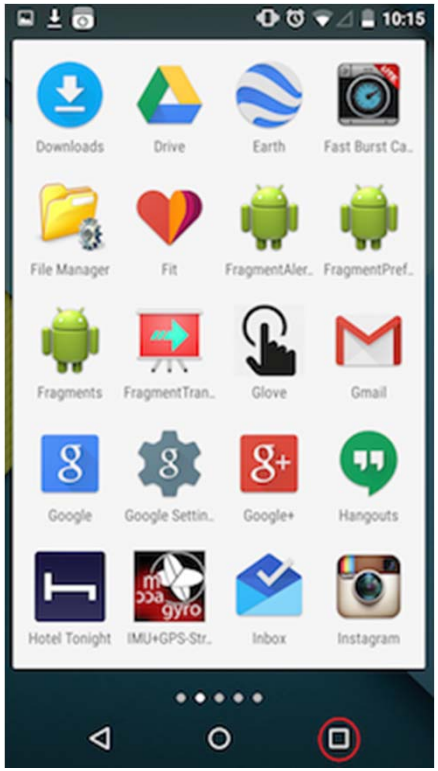


Android 5.0

all apps button



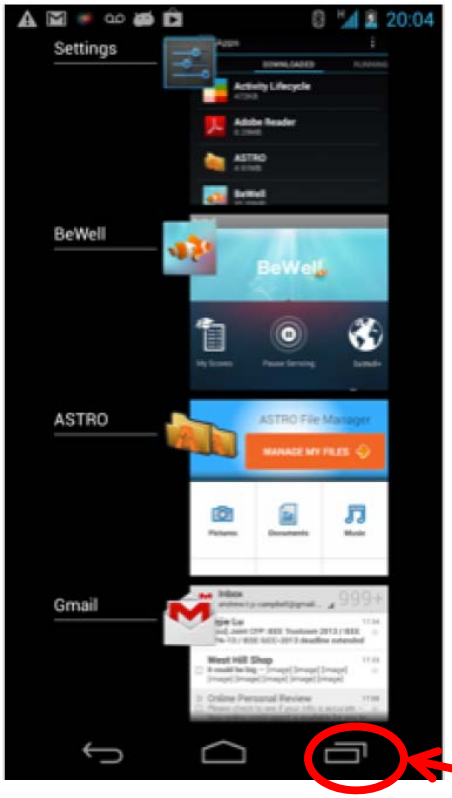
Android 4.0



Android 5.0

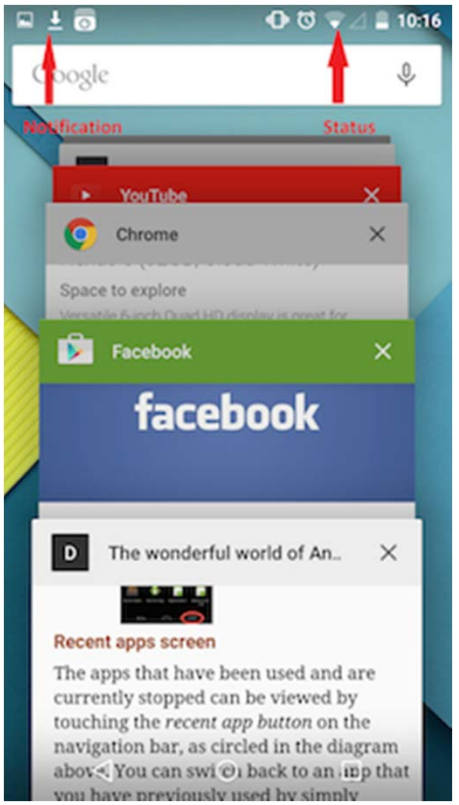
Recent Apps Screen

- Accessed by touching the **recent apps button**
- Shows recently used and currently stopped apps
- Can switch to a recently used app by touching it in list



Android 4.0

recent apps button

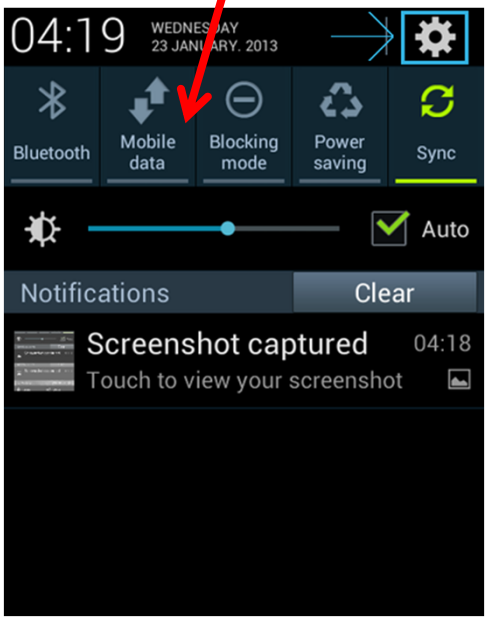


Android 5.0

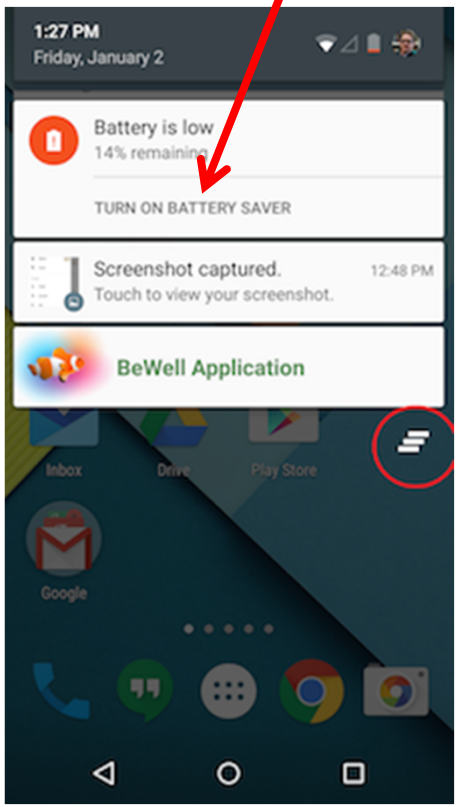
Status Bar and Notification Screen

- **Status:** time, battery, cell signal strength, bluetooth enabled, etc
- **Notification:** wifi, mail, bewell, voicemail, usb active, music, etc

Status bar



Notification Screen



Facebook UI

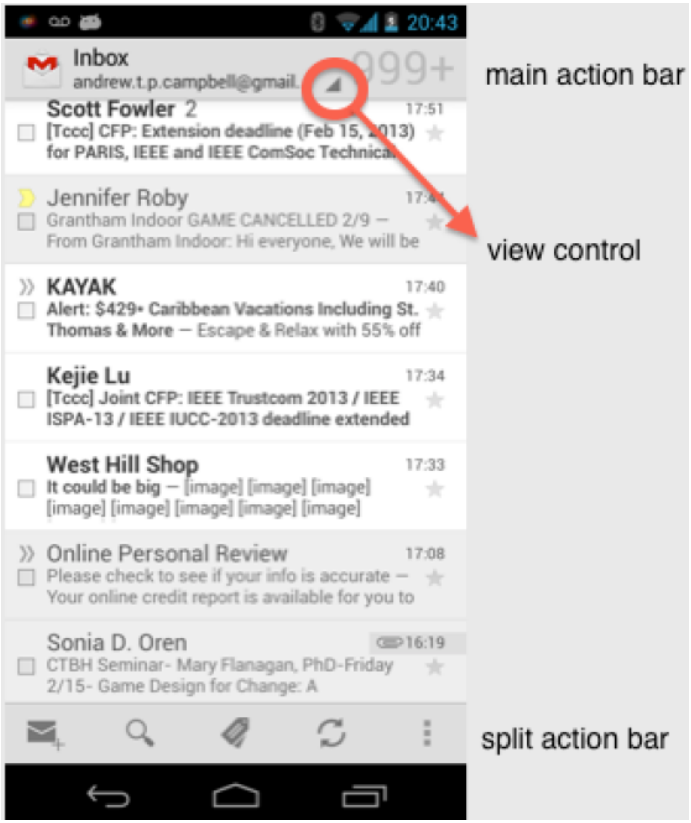
- Uses many standard Android UI components
- Shows **main action bar** and **split action bar**
- **Action bar:** configurable, handles user action, app navigation
- Split action bar at bottom of screen

Android 4.0

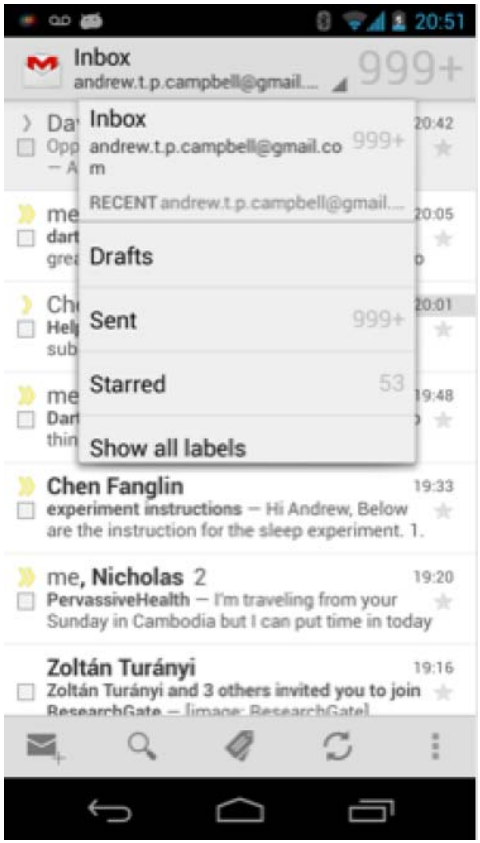
Android 5.0

Gmail

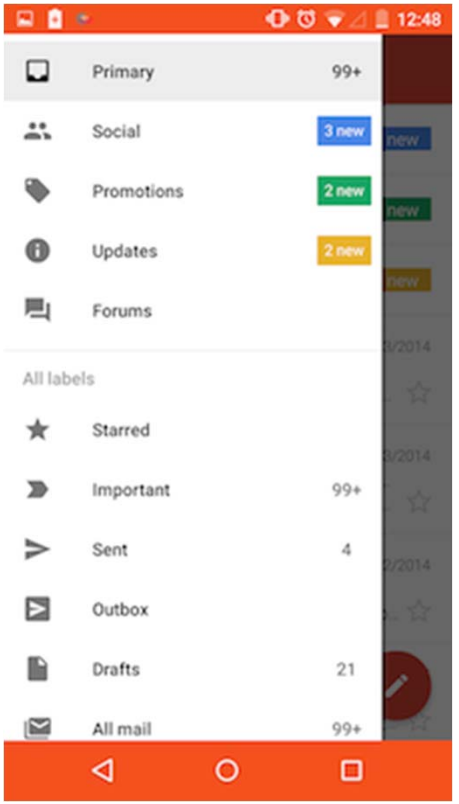
- **View control (pulldown menu):** allows users switch between different views (inbox, recent, drafts, sent)



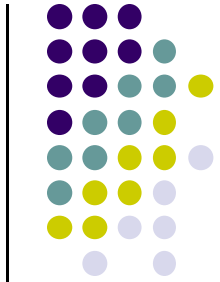
Android 4.0



Android 4.0

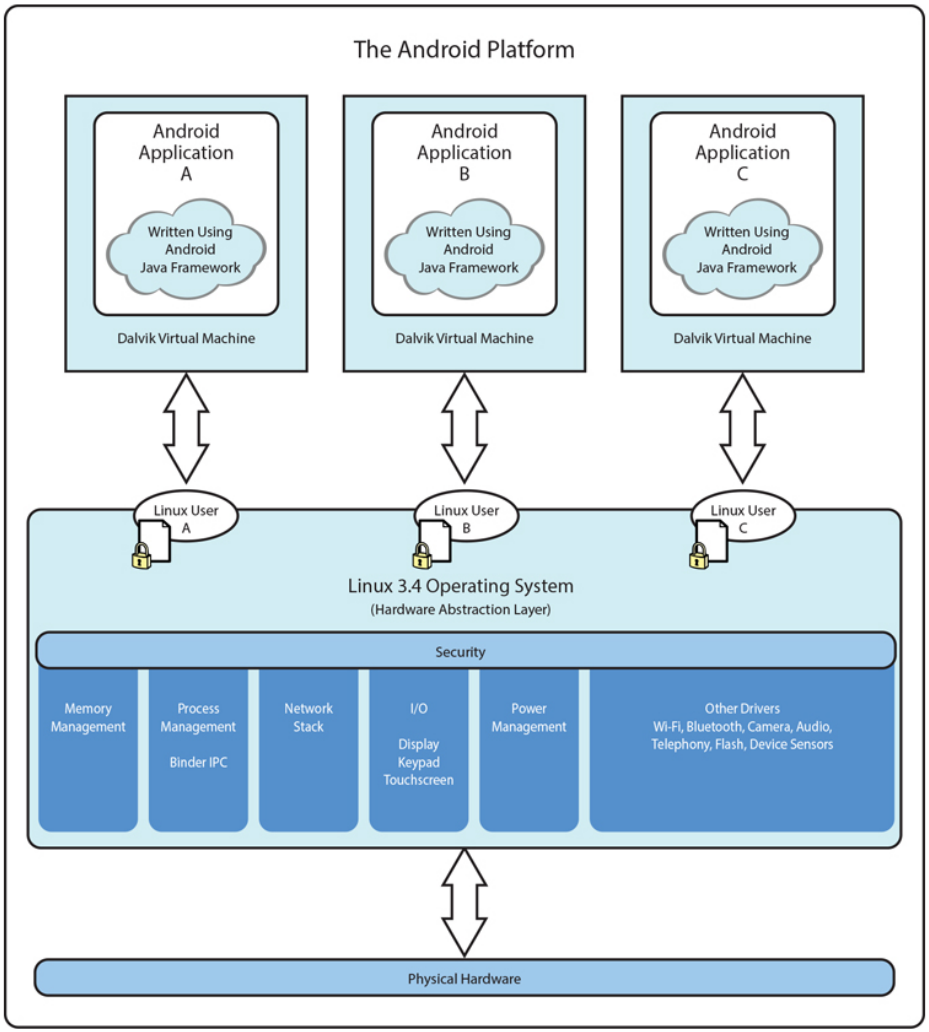


Android 5.0



Android Software Framework

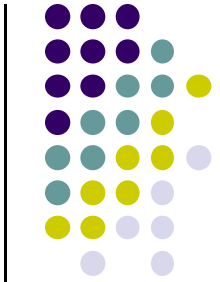
Android Software Framework



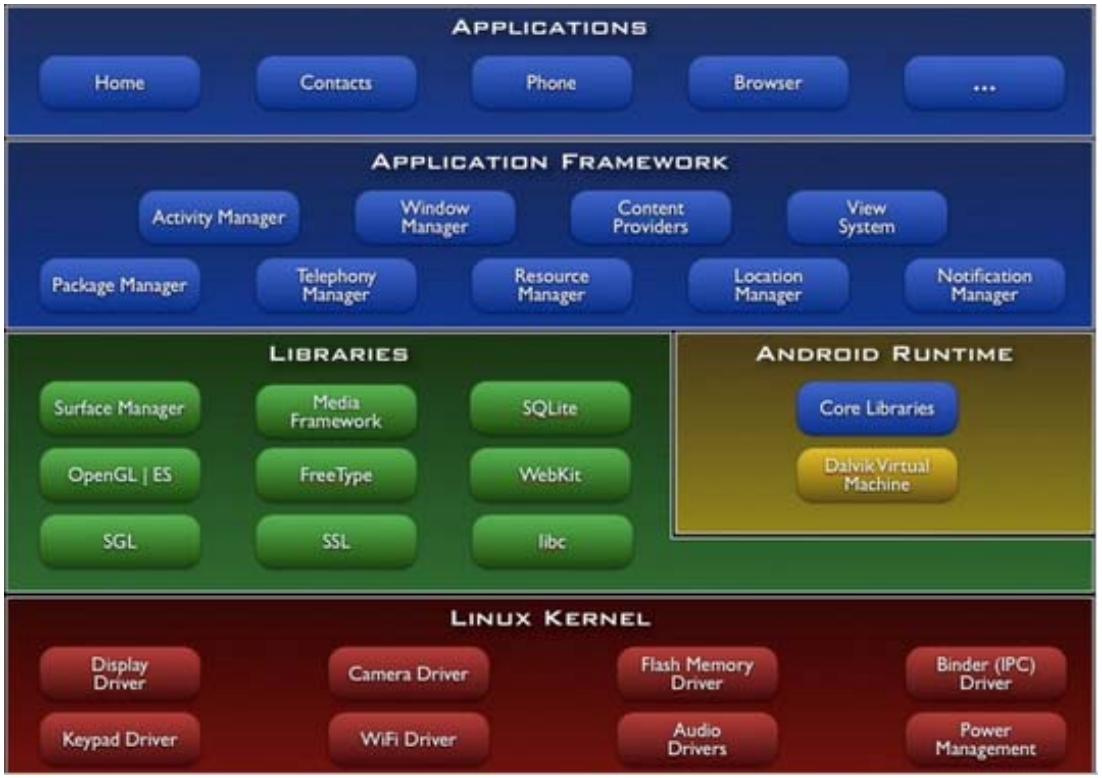
- Each Android app runs in its own security sandbox (VM, minimizes complete system crashes)
- Android OS multi-user Linux system
- Each app is a different user
- Application's files are private
- Android starts app's process when its components need to be executed, shuts down the process when no longer needed

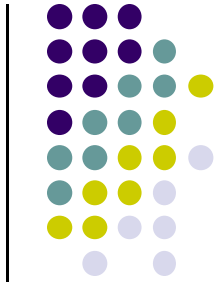
Ref: Introduction to Android Programming, Annuzzi, Darcey & Conder

Android Software Framework



- Android system assigns each app a unique Linux user ID
  - ID is unknown to the application
- Android system sets permissions for all an app's files so that only the process with the app's user ID can access them

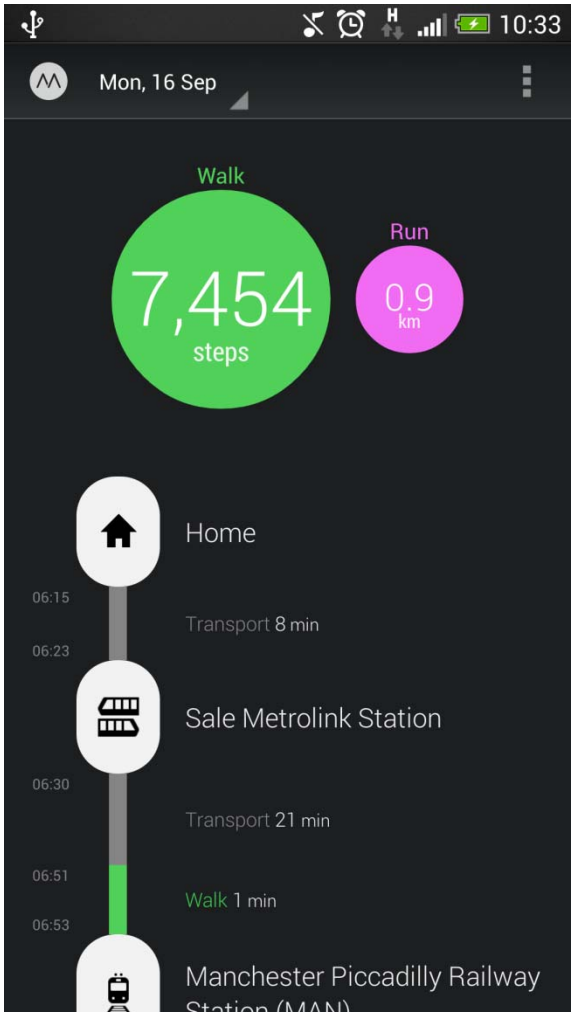




Android Files

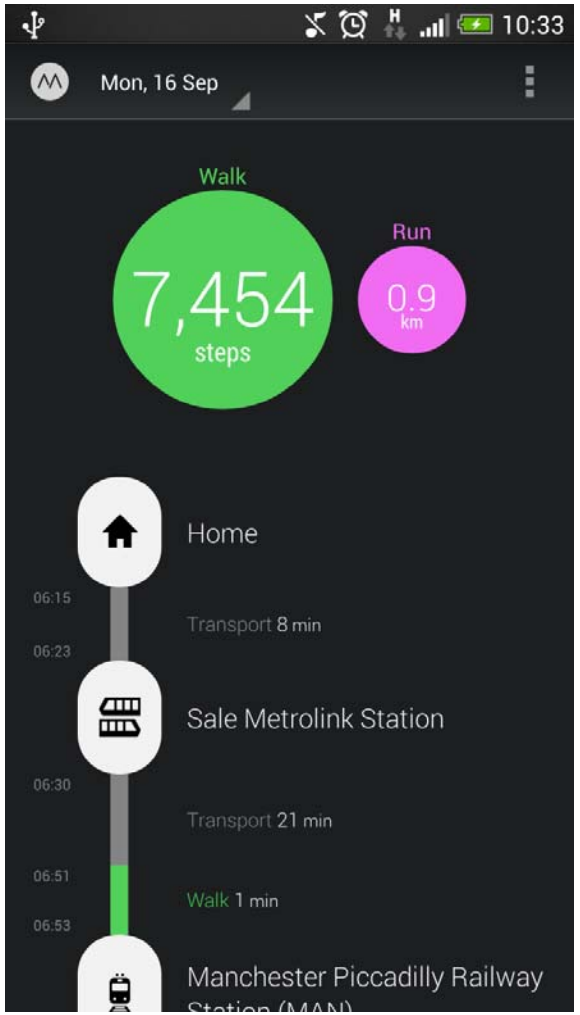
Android App

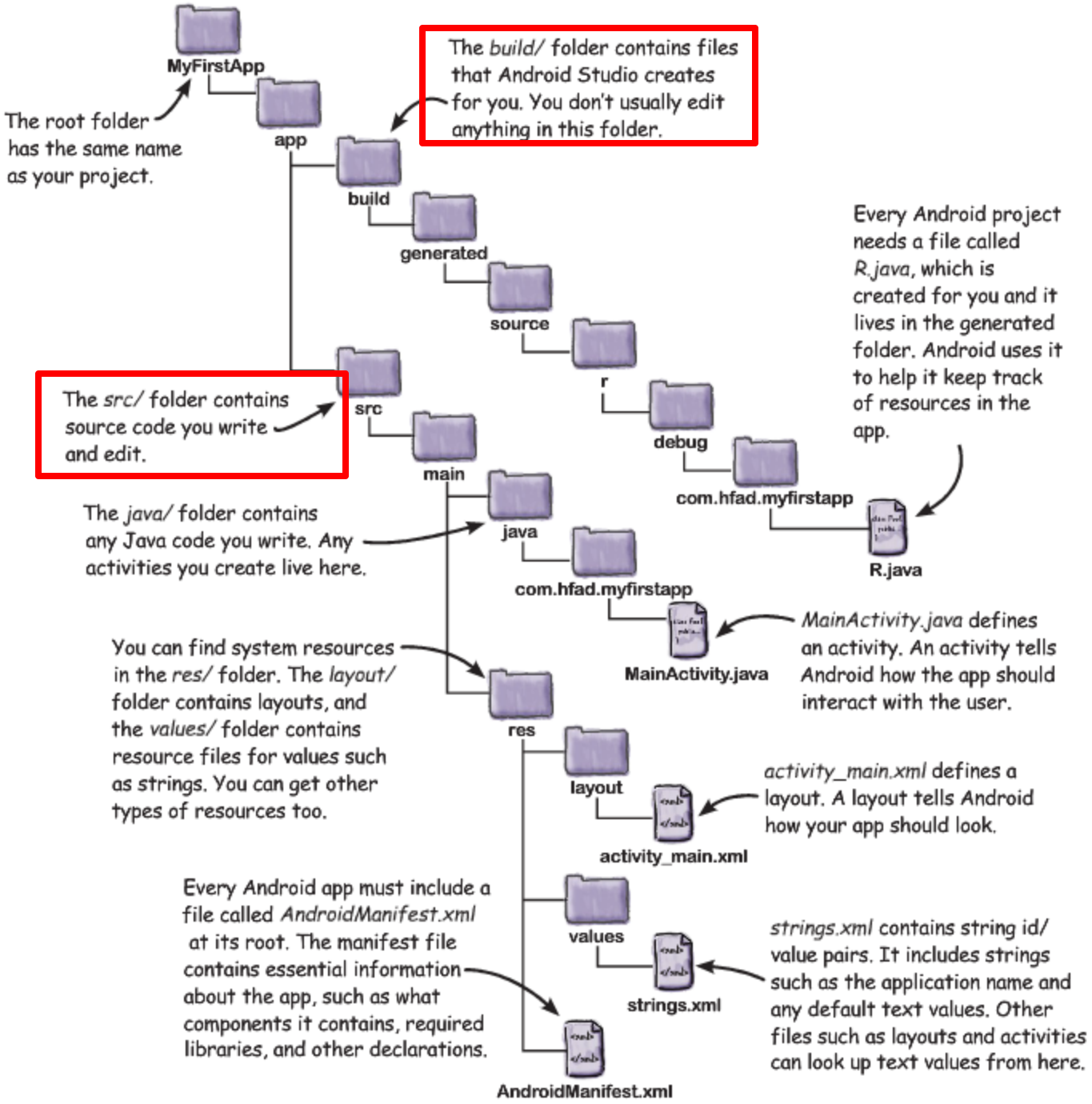
- Android apps written in Java
- Android SDK tools compile code, data and resource files into **Android PacKage (filename.apk)**.
  - .apk is similar to .exe on Windows
- Apps download from Google Play, or copied to device as **filename.apk**
- Installation = installing **apk file**
- App elements
  - User Interface
  - Other code running in background



UI Design using XML

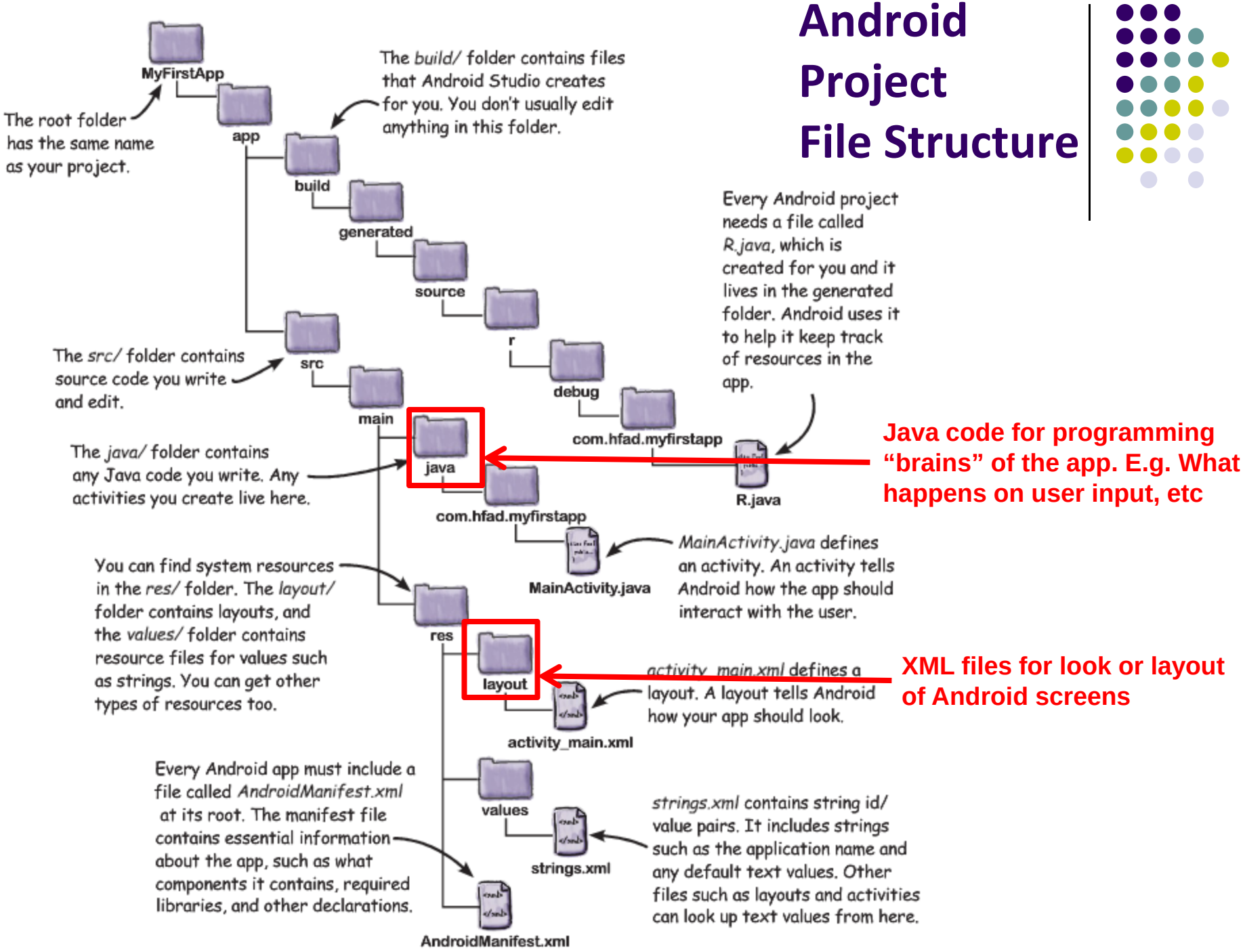
- Android separates UI design code from the program
- Why? UI can be modified without changing program, Java code
- **Example:** Shapes, colors can be changed in XML file without changing Java program
- UI designed using either:
  - Drag-and drop graphical (WYSIWYG) tool or
  - Programming Extensible Markup Language (XML)
- **XML:** Markup language, both human-readable and machine-readable''





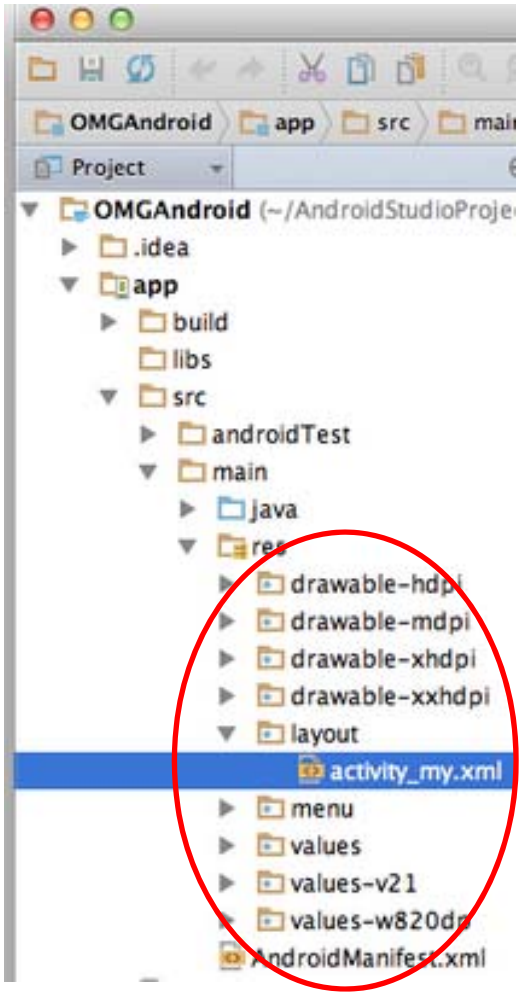
Android Project File Structure

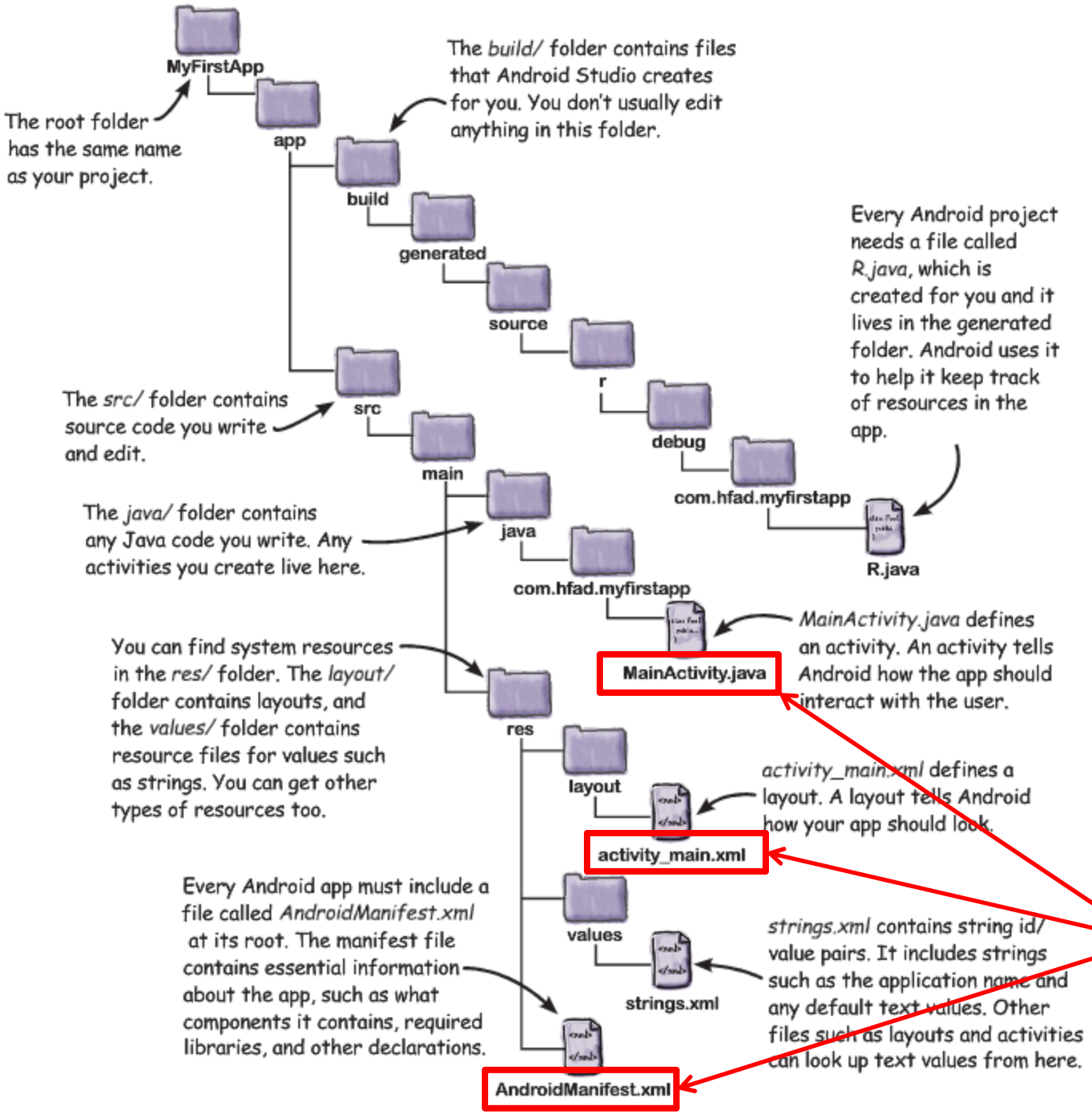
Android Project File Structure



Files in an Android Project

- **res/** folder contains resources you can embed in Android screen
- **res/menu/:** XML files for menu specs
- **res/drawable-xyz/:** images (PNG, JPEG, etc) at various resolutions
- **res/raw:** general-purpose files (e.g. audio clips, CSV files
- **res/values/:** strings, dimensions, etc

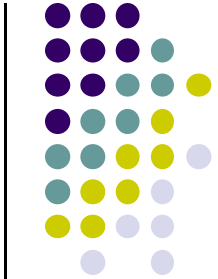




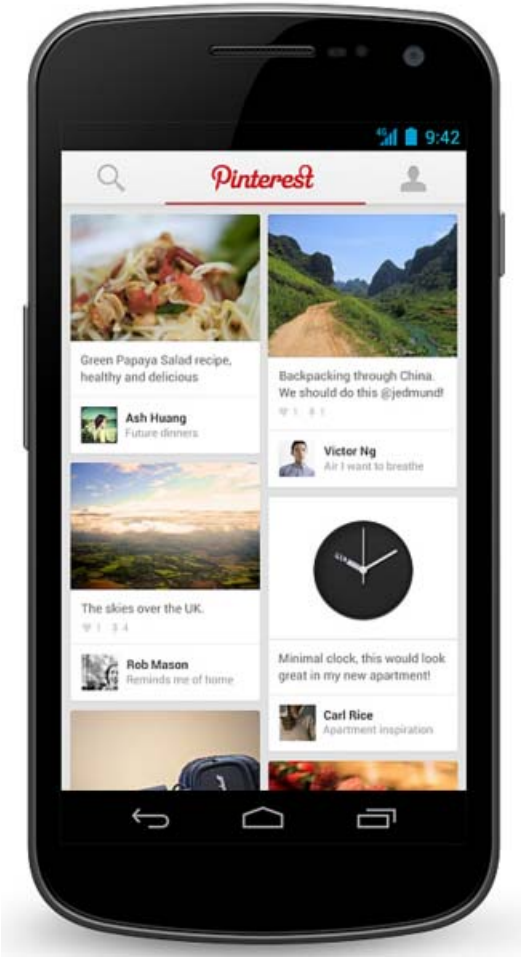
Android Project File Structure

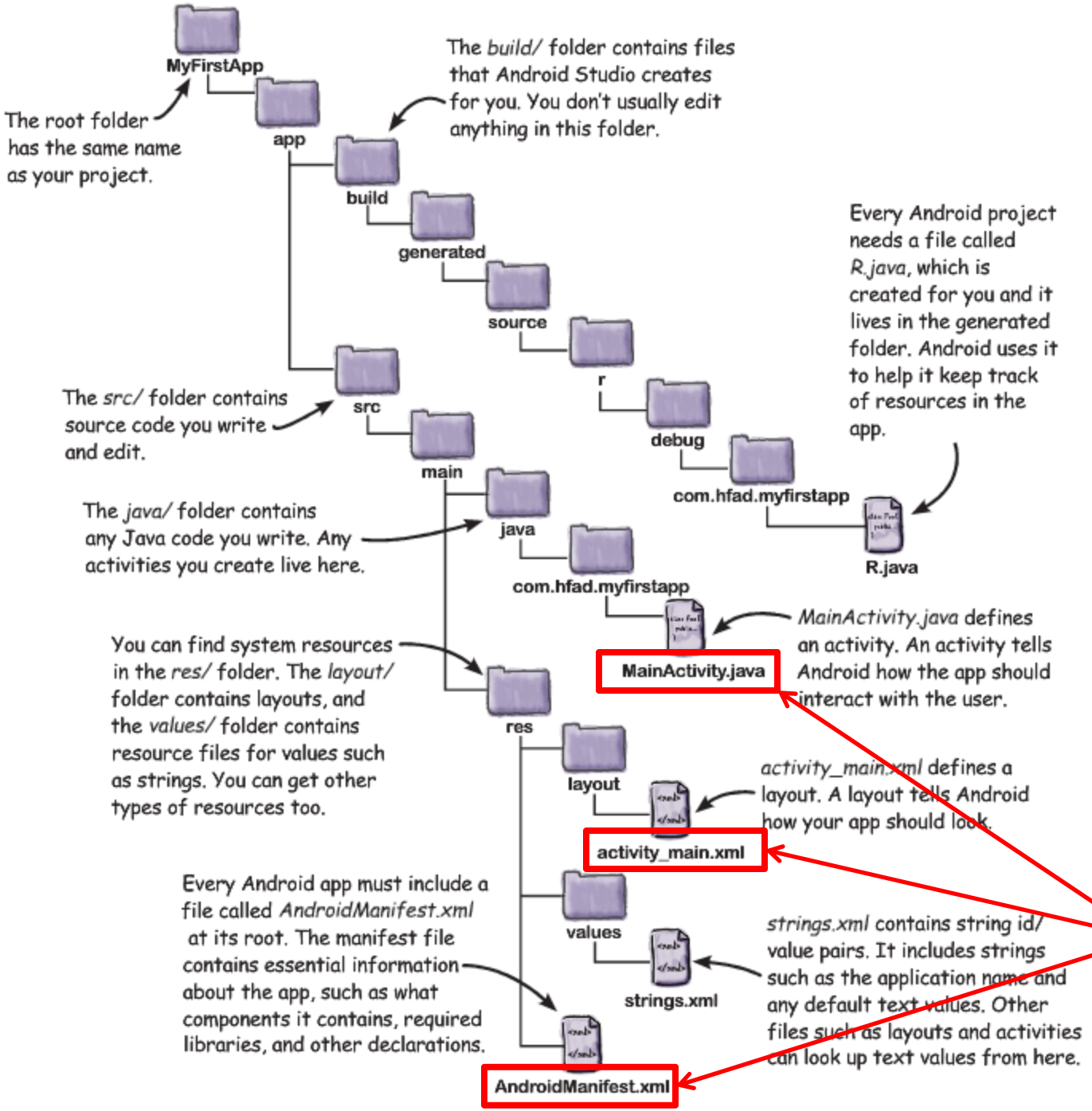
3 Main Files to Write Android app

Concrete Example: Files in an Android Project



- **res/layout:** layout, dimensions (width, height) of screen cells are specified in XML file here
- **res/drawable-xyz/:** The images stored in jpg or other format here
- **java/:** App's behavior when user clicks on a selection in java file here
- **AndroidManifext.XML:** Contains app name (Pinterest), list of app screens, etc



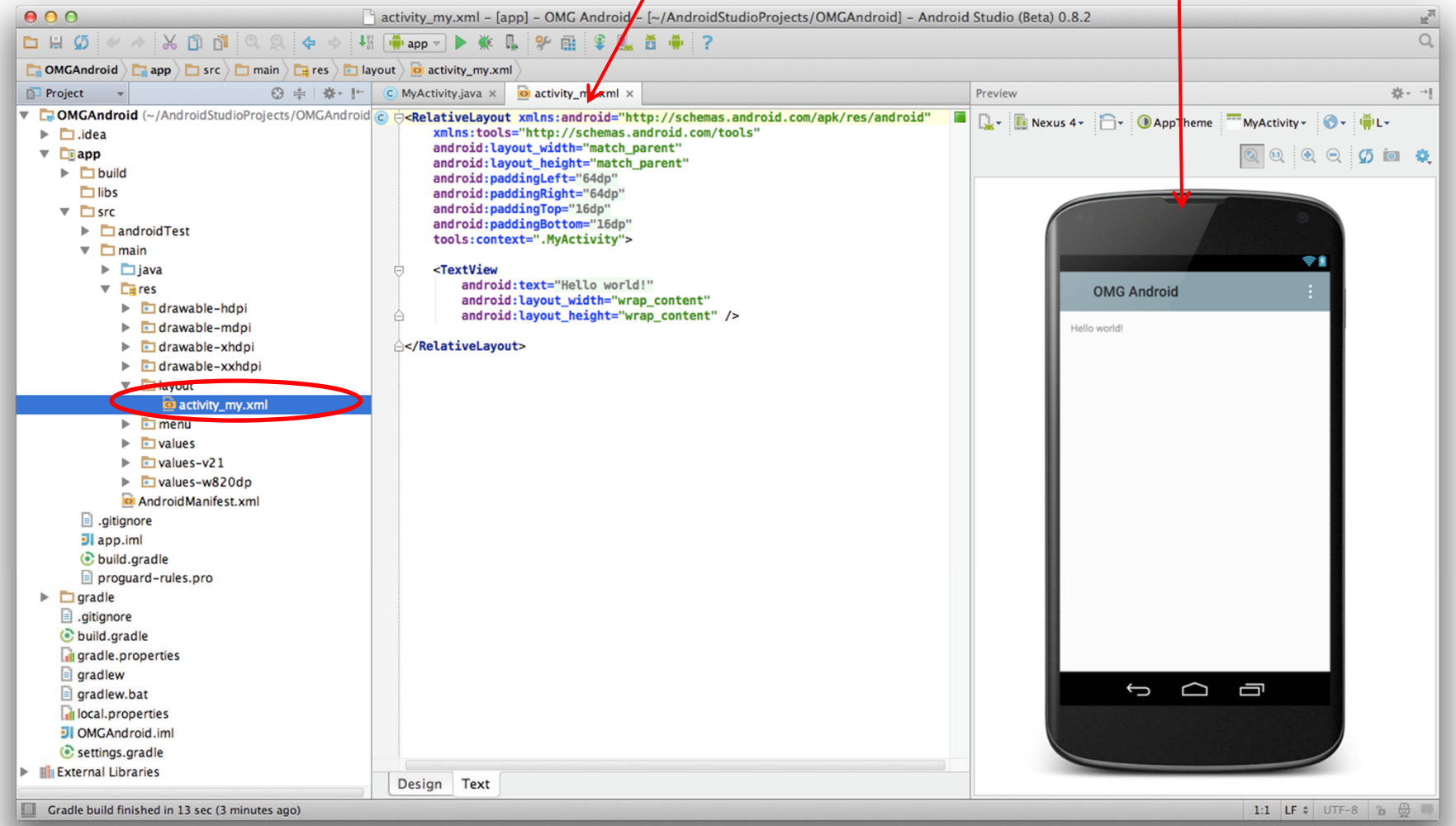


Recall: Android Project File Structure

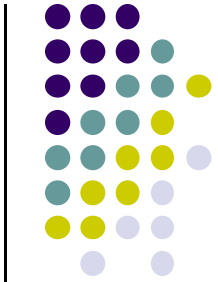
3 Main Files to Write Android app

Editting Android

- Activity_my.xml is XML file specifying screen layout
- Can edit XML directly or drag and drop



Basic Overview of an App

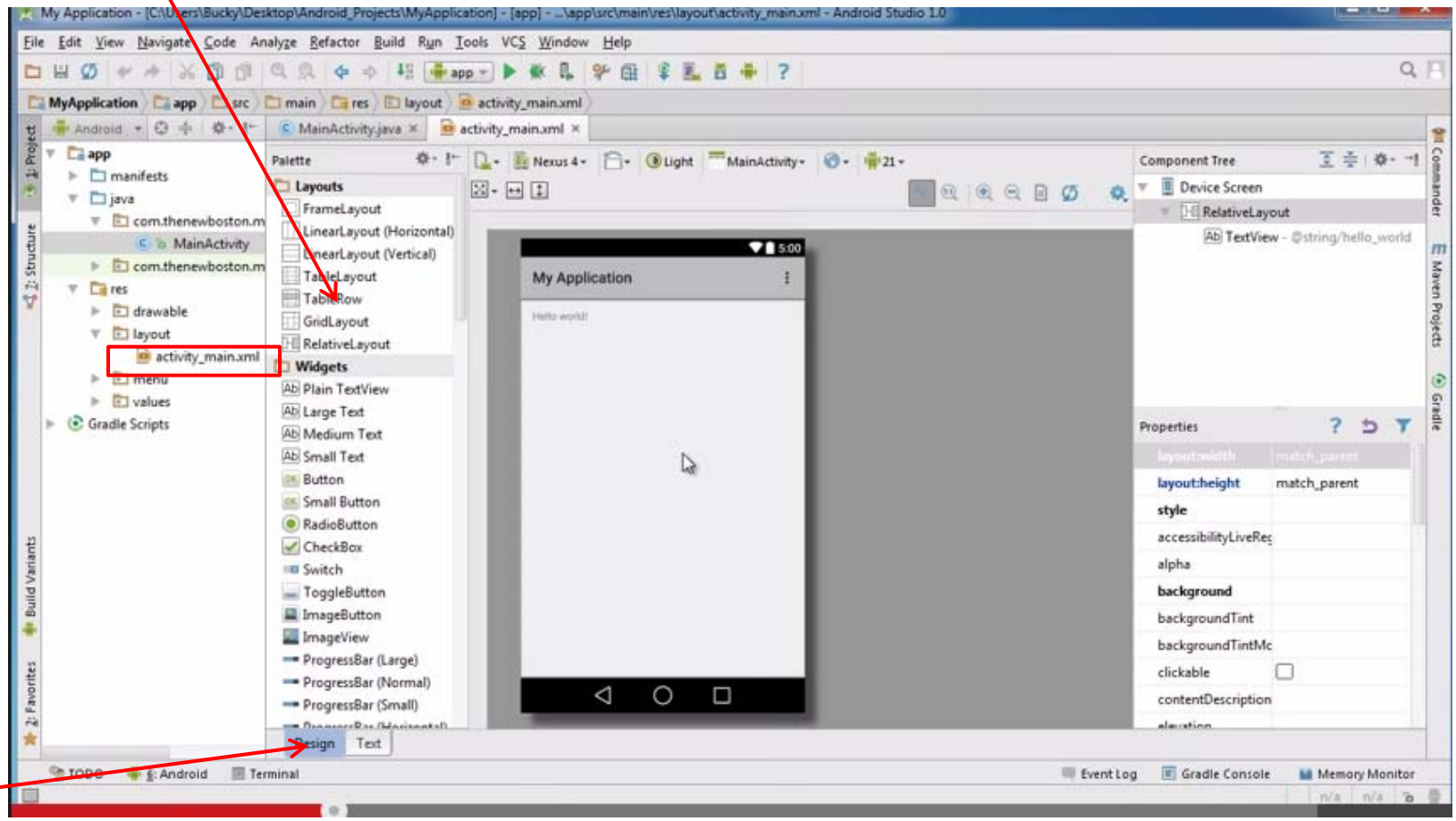


- Tutorial 8: Basic Overview of an App [11:36 mins]
  - https://www.youtube.com/watch?v=9l1lfWAiHPg

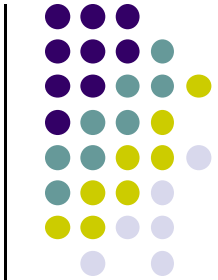
- Main topics
  - Introduces main files of Android App
    - Activity_main.xml
    - MainActivity.java
    - AndroidManifest.xml
  - How to work with these files within Android Studio
  - Editting files using either drag-and-drop interface or XML
  - Flow of basic app

Activity_main.xml

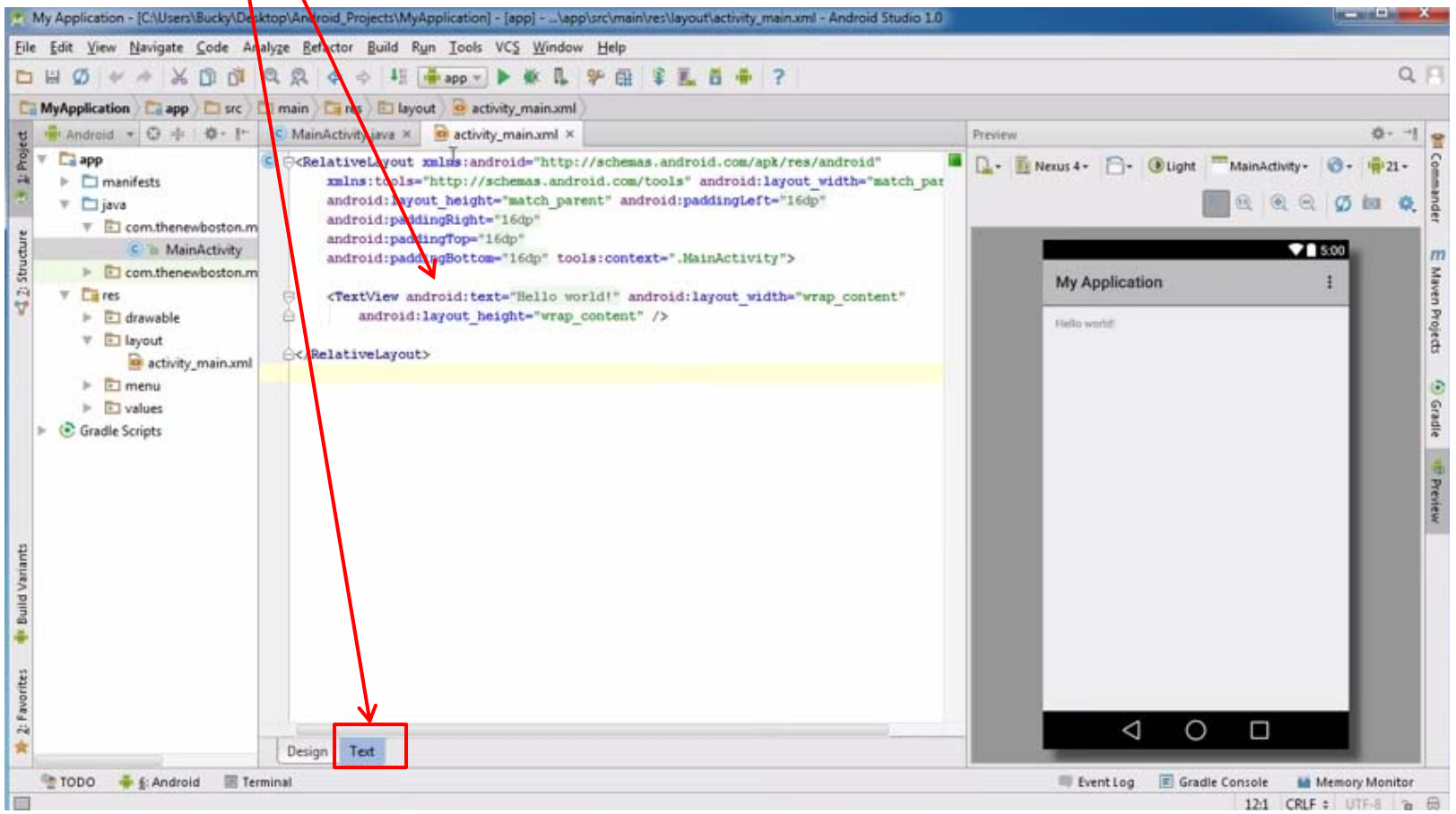
- XML file used to design screen layout, buttons, etc
- **Widgets:** elements that can be dragged onto activity (screen)
- **Design View:** Design app screen using Drag-and-drop widgets



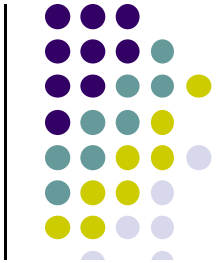
Activity_main.xml: Text View



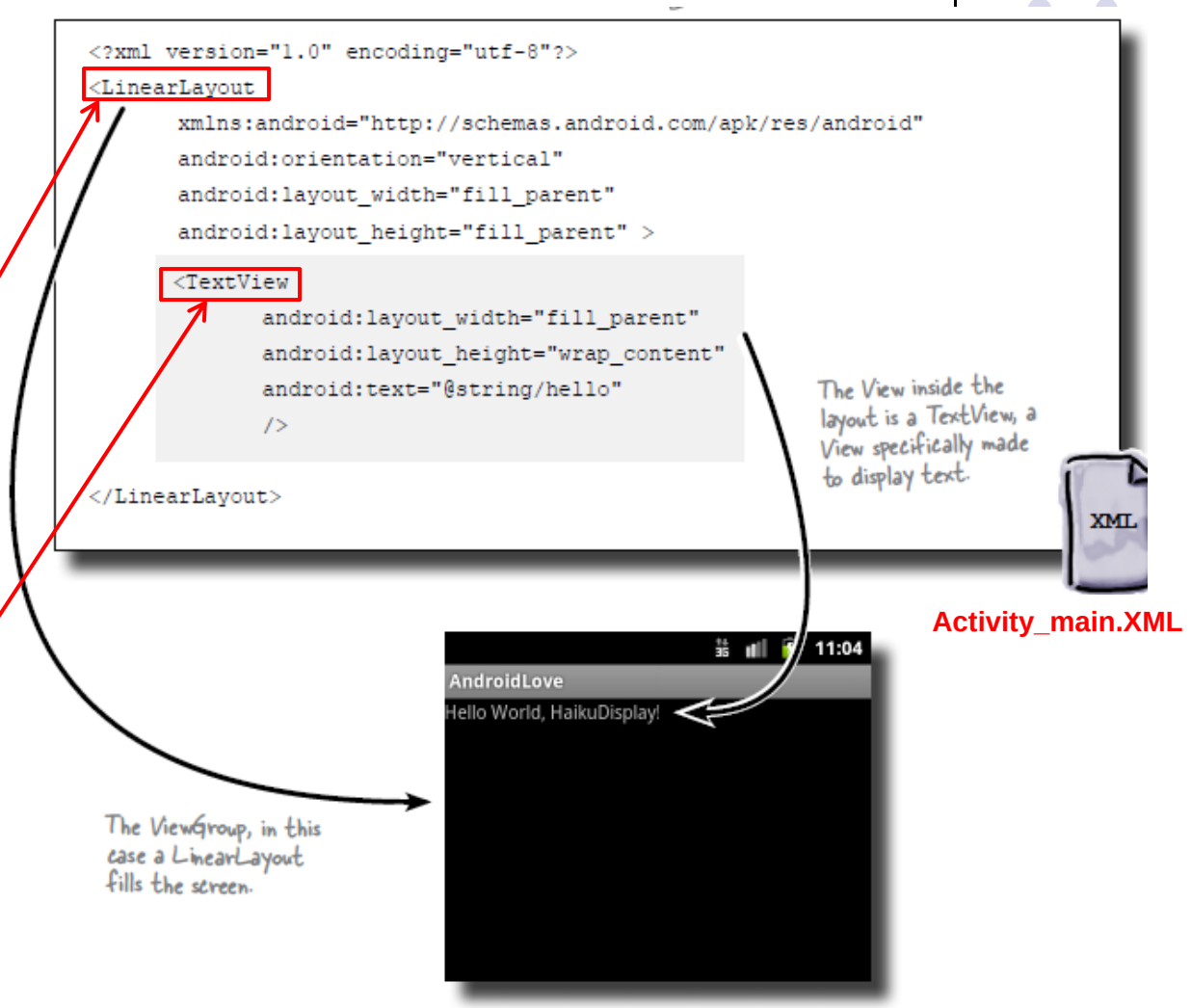
- **Text view:** Design screen by editting XML file directly
- **Note:** dragging and dropping widgets auto-generates corresponding XML



What's in XML Layout File (E.g. Activity_main.xml)?

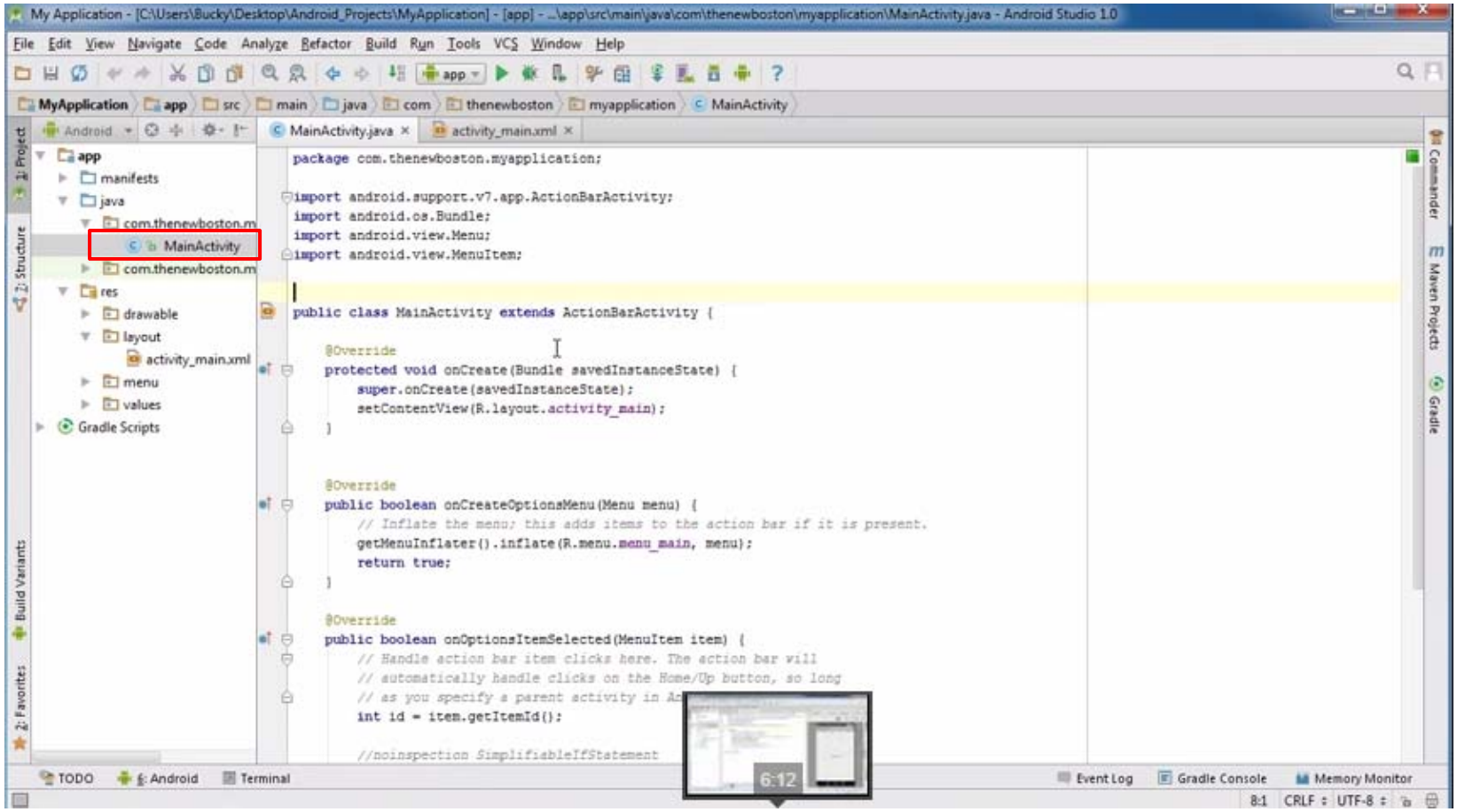


- XML Layout files consist of:
  - UI components (boxes) called **Views**
  - Different types of views. E.g
    - **TextView:** contains text,
    - **ImageView:** picture,
    - **WebView:** web page
  - **Views** arranged into layouts or **ViewGroups**
- Example XML file shown contains:
  - 1 ViewGroup (LinearLayout) that fills the entire screen
  - 1 View (TextView) that contains text



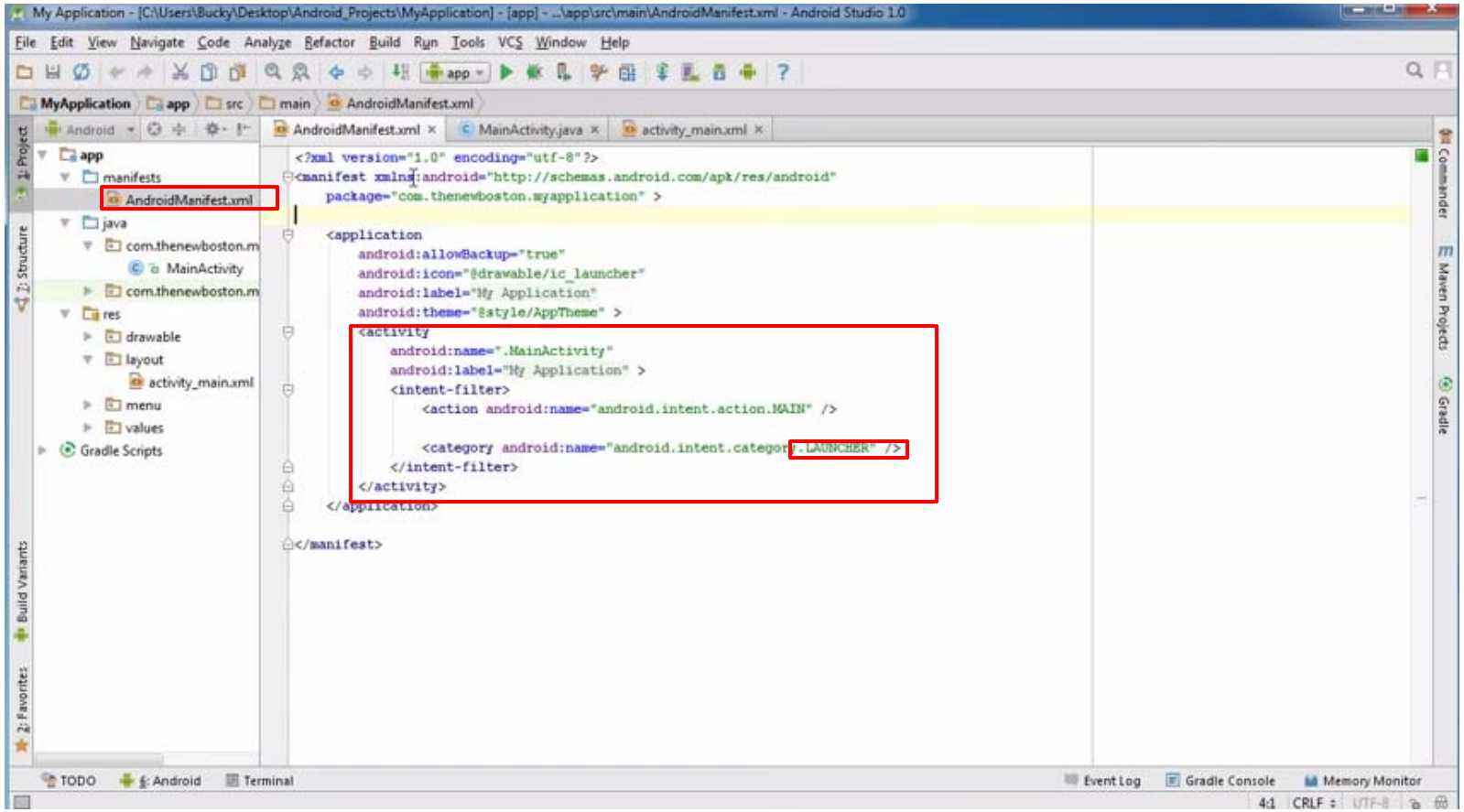
MainActivity.java

- Java code, defines actions, handles interaction/put taken (intelligence)
  - E.g. What app will do when button/screen clicked

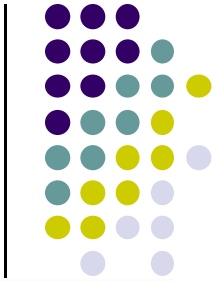


AndroidManifest.xml

- App's starting point (a bit like main( ) in C)
- All app screens (activities) are listed in AndroidManifest.xml
- Activity with tag "LAUNCHER" is launched first (starting point)



Inside “Hello World” AndroidManifest.xml



Your package name

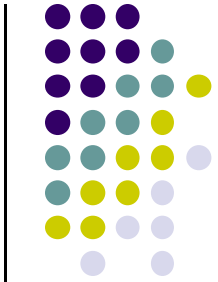
Android version

List of activities (screens) in your app

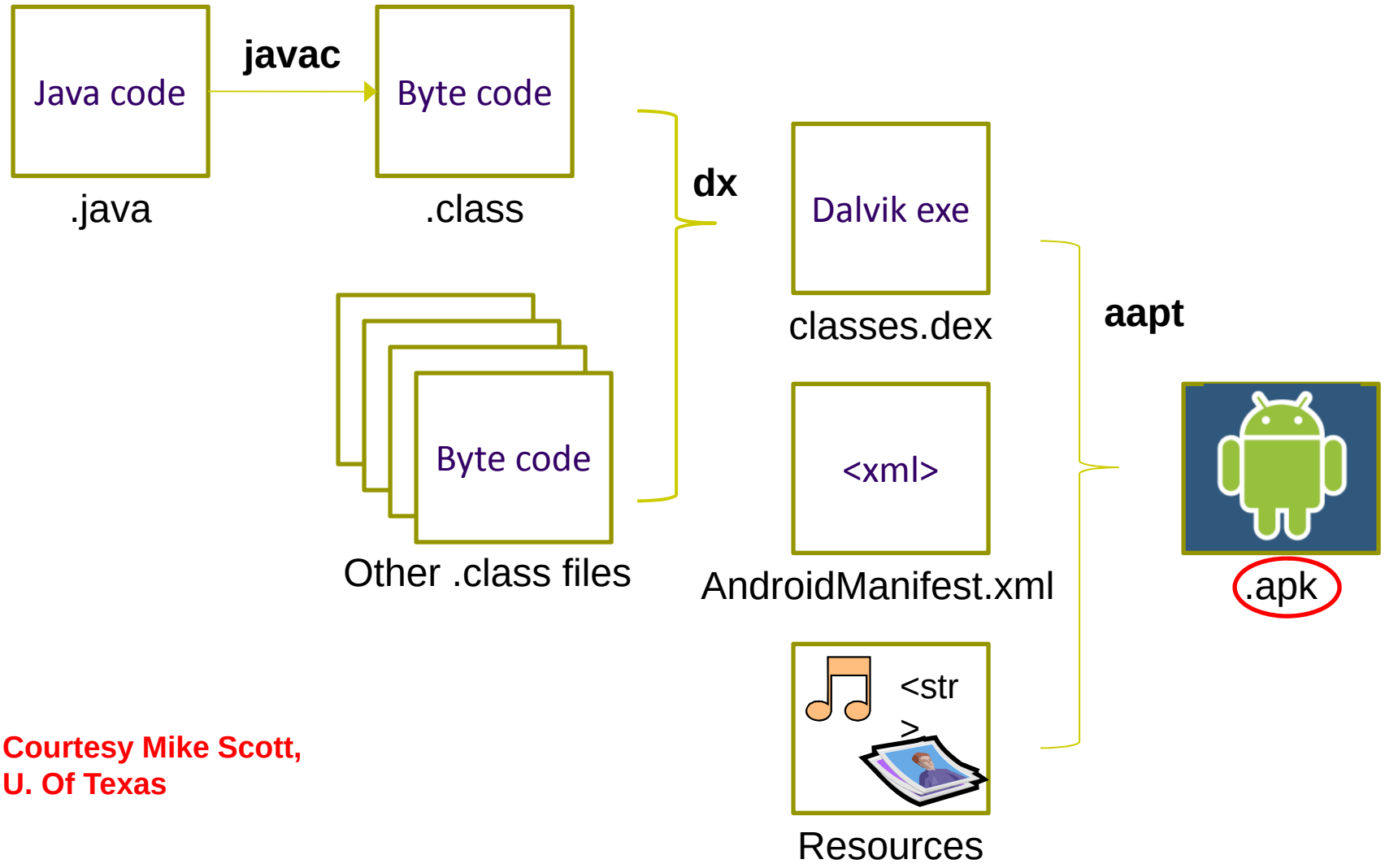
```
<?xml version="1.0"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.skeleton"
  android:versionCode="1"
  android:versionName="1.0">

  <application>
    <activity
      android:name="Now"
      android:label="Now">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

One activity (screen) designated LAUNCHER. The app starts running here

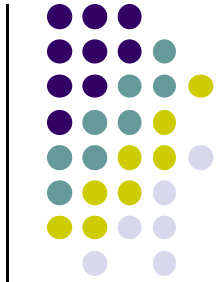
Android Compilation Process/Steps



- Dalvik is Android virtual machine
  - Works like Java virtual machine, but optimized for mobile devices



Courtesy Mike Scott,
U. Of Texas



Our First Android App

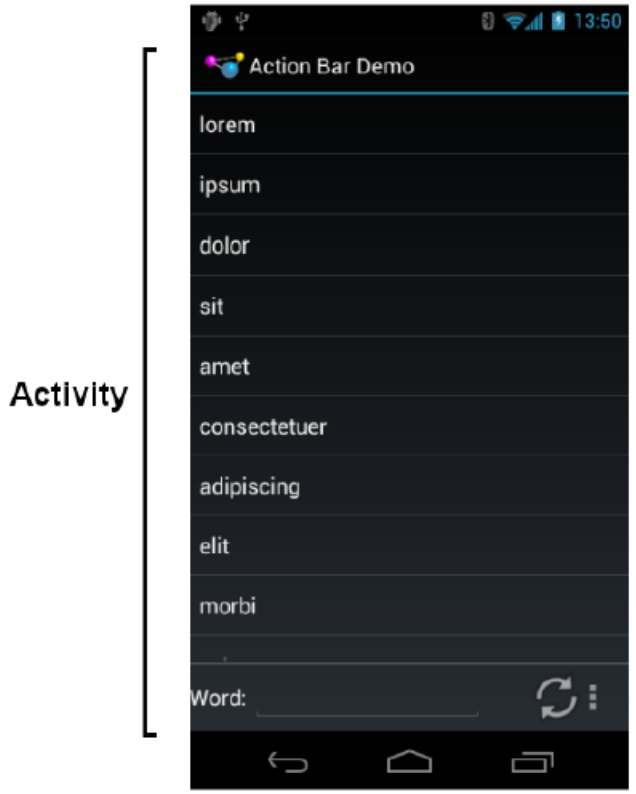
Activities

- Activity: 1 Android screen or dialog box
- Apps
  - Have at least 1 activity that deals with UI
  - Entry point of app similar to **main( )** in C
  - Typically have multiple activities

- Example: A camera app
  - **Activity 1:** to focus, take photo, launch activity 2
  - **Activity 2:** to view photo, save it

Activities

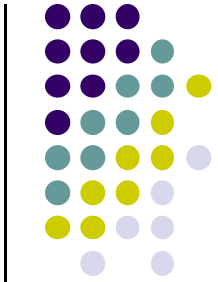
- Activities independent of each other
- Activity 1 can write data, read by activity 2
- App Activities are sub-class of **Activity** class



Recall: Files Hello World Android Project

- 3 Files:
  - **Activity_my.xml:** XML file specifying screen layout
  - **MainActivity.Java:** Java code to define behavior, actions taken when button clicked (intelligence)
  - **AndroidManifest.xml:**
    - Lists all screens, components of app
    - Analogous to a table of contents for a book
    - E.g. Hello world program has 1 screen, so AndroidManifest.xml has 1 item listed
    - App starts running here (a bit like main( ) in C), launches activity with a tag “LAUNCHER”

Execution Order



Start in AndroidManifest.xml
Read list of activities (screens)
Start execution from Activity tagged Launcher

Create/execute activities (declared in java files)
E.g. MainActivity.Java

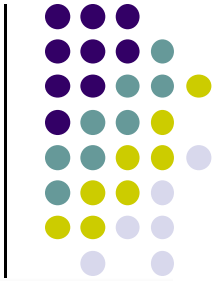
Format each activity using layout In XML file (e.g. Activity_my.xml)

Recall: Files Hello World Android Project

- 3 Files:
  - **Activity_my.xml:** XML file specifying screen layout
  - **MainActivity.Java:** Java code to define behavior, actions taken when button clicked (intelligence)
  - **AndroidManifest.xml:**
    - Lists all screens, components of app
    - Analogous to a table of contents for a book
    - E.g. Hello world program has 1 screen, so AndroidManifest.xml has 1 item listed
    - App starts running here (a bit like main( ) in C), launching activity with a tag “LAUNCHER”

Already saw AndroidManifest.XML

Recall: Inside "Hello World" AndroidManifest.xml



Your package name

Android version

List of activities (screens) in your app

```
<?xml version="1.0"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.skeleton"
  android:versionCode="1"
  android:versionName="1.0">

  <application>
    <activity
      android:name="Now"
      android:label="Now">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

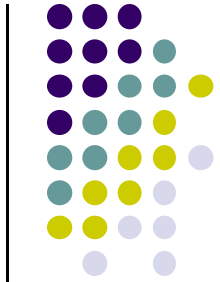
One activity (screen) designated LAUNCHER. The app starts running here

Recall: Files Hello World Android Project

- 3 Files:
  - **Activity_my.xml:** XML file specifying screen layout
  - **MainActivity.Java:** Java code to define behavior, actions taken when button clicked (intelligence)
  - **AndroidManifest.xml:**
    - Lists all screens, components of app
    - Analogous to a table of contents for a book
    - E.g. Hello world program has 1 screen, so AndroidManifest.xml has 1 item listed
    - App starts running here (a bit like main( ) in C), launching activity with a tag "LAUNCHER"

Next: Let's look at Simple java file

Example Activity Java file (E.g. MainActivity.java)



Package declaration (Same as chosen initially)

Import needed classes

My class inherits from Android activity class

Initialize by calling onCreate( ) method of base Activity class

```
package com.commonsware.empublite;

import android.app.Activity;
import android.os.Bundle;

public class EmPubLiteActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
}
```

Use screen layout (design) declared in file main.xml stored in folder res/layout

Note: Android calls your Activity's onCreate method once it is created

Recall: Files Hello World Android Project

XML file used to design Android UI

- 3 Files:
  - **Activity_my.xml:** XML file specifying screen layout
  - **MainActivity.Java:** Java code to define behavior, actions taken when button clicked (intelligence)
  - **AndroidManifest.xml:**
    - Lists all screens, components of app
    - Analogous to a table of contents for a book
    - E.g. Hello world program has 1 screen, so AndroidManifest.xml has 1 item listed
    - App starts running here (a bit like main( ) in C), launching activity with a tag “LAUNCHER”

Simple XML file Designing UI

- After choosing the layout, then widgets added to design UI

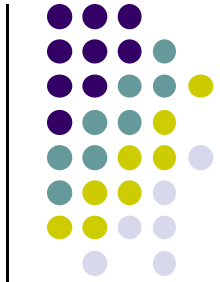
This file is written using xml namespace and tags and rules for android

```
<RelativeLayout 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"
   tools:context=".EmPubLiteActivity">

   <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_centerHorizontal="true"
     android:layout_centerVertical="true"
     android:text="@string/hello_world"/>

</RelativeLayout>
```

Declare Layout

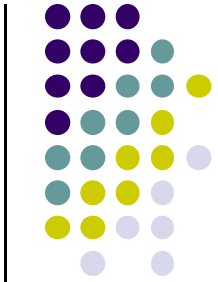
Add widgets

Widget properties
(e.g. center contents
horizontally and vertically)



Resources

View Properties and String Resources



- Views are declared with attributes for configuring them
- Consider the following TextView example

```
<TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
```

These XML properties define the width and height of the view.

This attribute sets the text on the view.

- **@string/hello** is a variable declared in another file, **strings.xml**

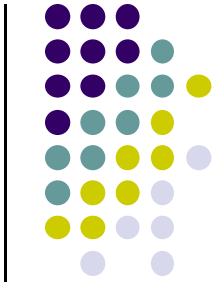
The raw XML showing name/value strings resources.

main.xml | strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HaikuDisplay!</string>
    <string name="app_name">AndroidLove</string>
</resources>
```

Resources | strings.xml

Strings in AndroidManifest.xml



- Strings declared in strings.xml can be referenced by all other XML files (activity_my.xml, AndroidManifest.xml)

String declaration in strings.xml

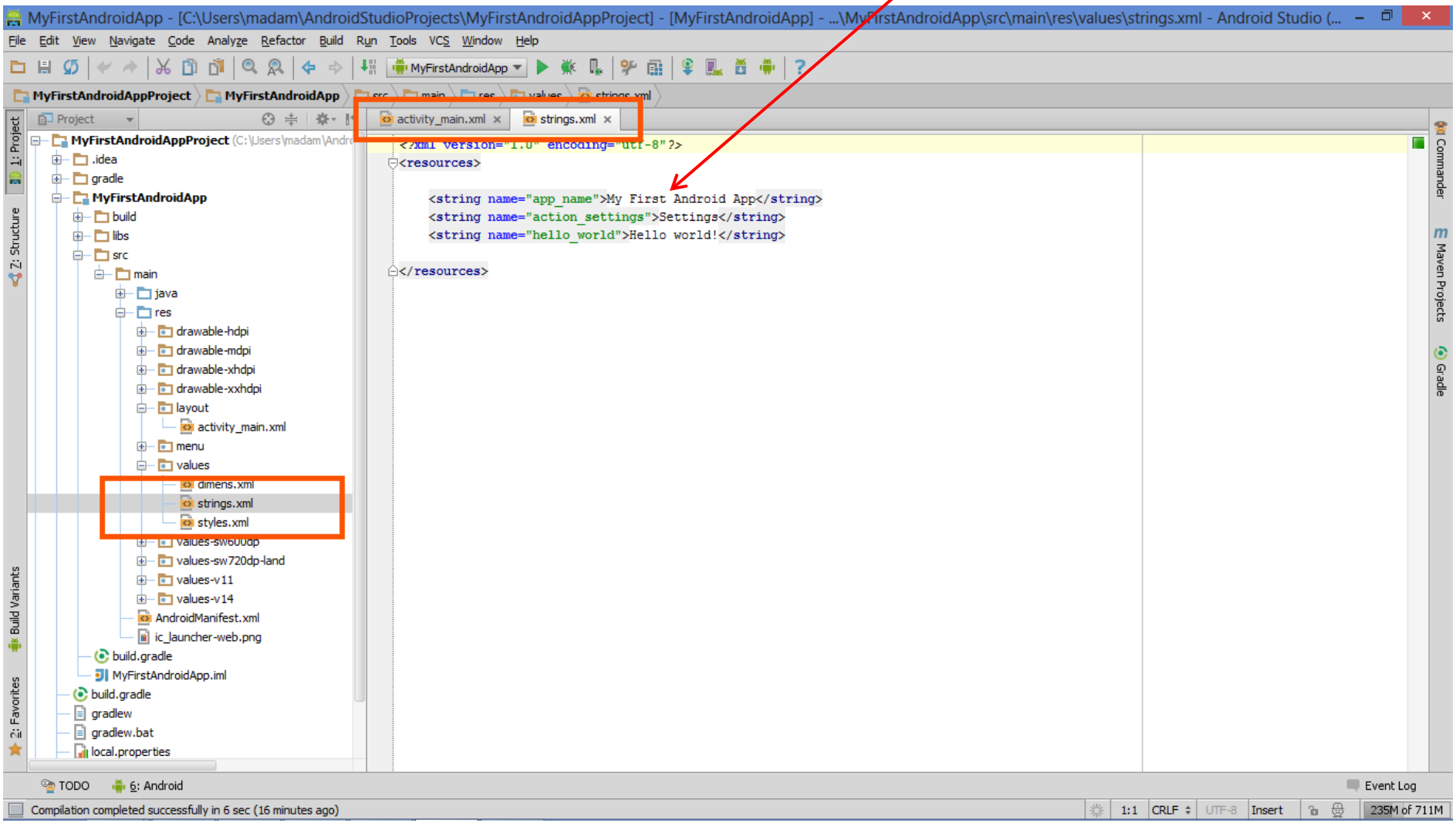
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="app_name">EmPubLite</string>
  <string name="hello_world">Hello world!</string>

</resources>
```

String usage in AndroidManifest.xml

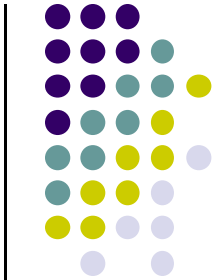
```
  <application
    android:allowBackup="false"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity
      android:name="EmPubLiteActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>

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

</manifest>
```

Where is strings.xml in Android Studio?

Editting any string in strings.xml changes it wherever it is displayed



Styled Text

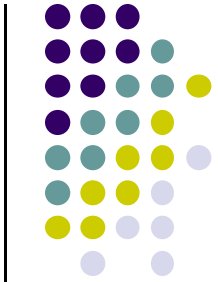


- In HTML, tags can be used for italics, bold, etc
  - E.g. Hello makes text *Hello*
  - <b> Hello <b> makes text **Hello**

- Can use the same HTML tags to add style (italics, bold, etc) to Android strings

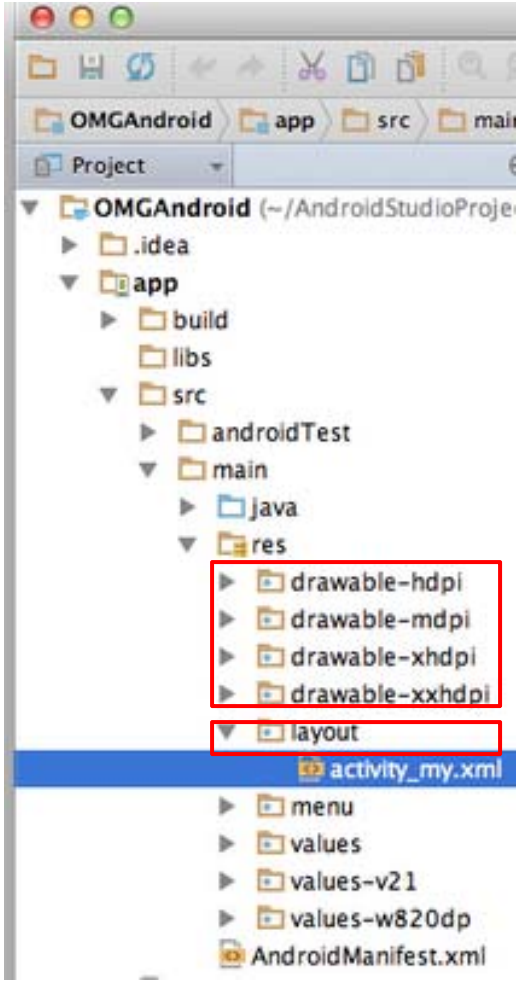
```
<resources>
  <string name="b">This has <b>bold</b> in it.</string>
  <string name="i">Whereas this has <i>italics</i>!</string>
</resources>
```

Recall: Example: Files in an Android Project



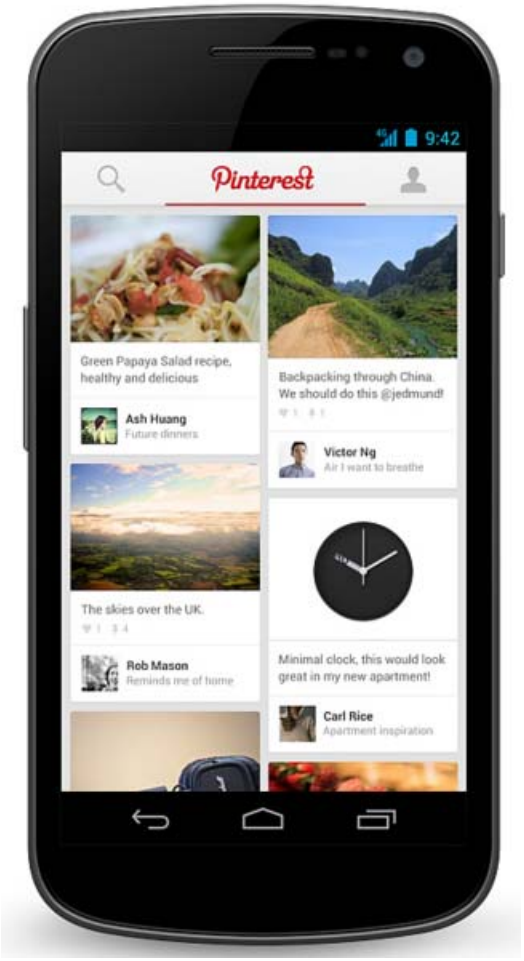
- **res/layout:** The width, height, layout of screen cells are specified in XML file here

- **res/drawable-xyz/:** The images stored in jpg or other format here

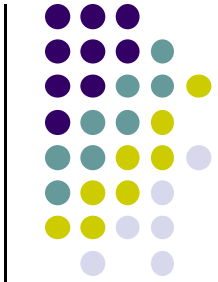


Resource Files in an Android Project

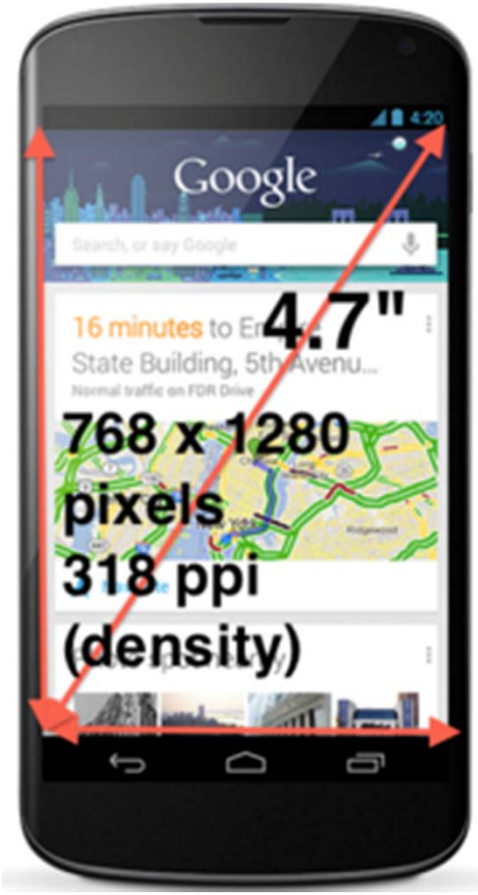
- Resources (stored in **/res** folder) are static bits of information outside java code (e.g. layout, images, etc). E.g.
  - **res/drawable-xyz/**
  - **res/layout:**
- Can have multiple resource definitions, used under different conditions. E.g internalization (text in different languages)
- In Android Studio, the **res/** folder is **app/src/main/**



Phone Dimensions Used in Android UI

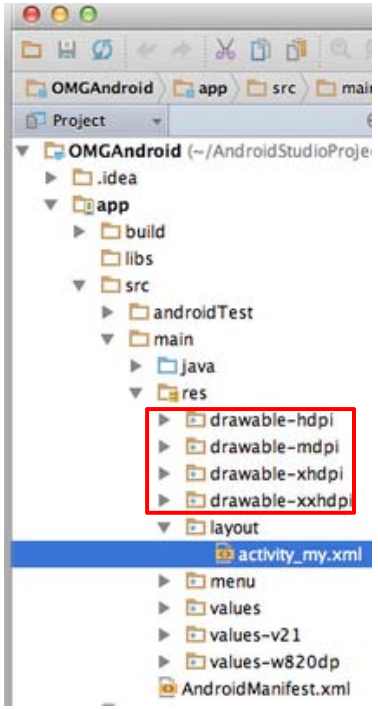


- Physical dimensions measured diagonally
  - E.g. Nexus 4 is 4.7 inches diagonally
- Resolution in pixels
  - E.g. Nexus 4 resolution 768 x 1280 pixels
- Pixels per inch (PPI) =
  - Sqrt[(768 x 768) + (1280 x 1280)] / 4.7= 318
- Dots per inch (DPI) is number of pixels in a physical area
  - Low density (ldpi) = 120 dpi
  - Medium density (mdpi) = 160 dpi
  - High density (hdpi) = 240 dpi
  - Extra High Density (xhdpi) = 320 dpi

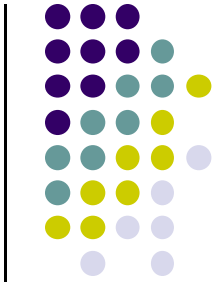


Adding Pictures

- Android supports images in PNG, JPEG and GIF formats
- GIF officially discouraged, PNG preferred format
- Default directory for images (drawables) is **res/drawable-xyz**
- Images in **res/drawable-xyz** can be referenced by XML and java files
  - **res/drawable-ldpi:** low dpi images (~ 120 dpi of dots per inch)
  - **res/drawable-mdpi:** medium dpi images (~ 160 dpi)
  - **res/drawable-hdpi:** high dpi images (~ 240 dpi)
  - **res/drawable-xhdpi:** extra high dpi images (~ 320 dpi)
  - **res/drawable-xxhdpi:** extra extra high dpi images (~ 480 dpi)
  - **res/drawable-xxxhdpi:** high dpi images (~ 640 dpi)
- Images in these directories are different resolutions, same size



Adding Pictures

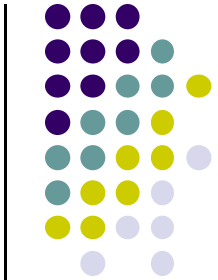


- Just the generic picture name is used (no format e.g. png)
  - No specification of what resolution to use
  - E.g. to reference an image **ic_launcher.png**

```
<application
  android:allowBackup="false"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme">
```

- Android chooses which directory (e.g. –mdpi) at run-time based on actual device resolution
- Android studio tools for generating icons
  - **Icon wizard or Android asset studio:** generates icons in various densities from starter image
  - Cannot edit images (e.g. dimensions) with these tools

Editting Pictures

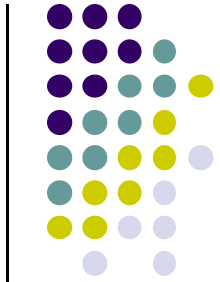


- **Image dimensions:**
  - **px:** hardware pixels, varies from device to device
  - **in** and **mm:** inches or millimeters
  - **pt:** 1/72nd of an inch
  - **dip (**or **dp):** density-independent pixels
    - 1 dip = 1 hardware pixel on ~160 dpi screen
    - 1 dip = 2 hardware pixels on ~ 320 dpi screen
  - **sp** (or scaled pixels)**:** scaled pixels
- Dimensions declared in **dimens.xml**

```
<resources>
  <dimen name="thin">10dip</dimen>
  <dimen name="fat">1in</dimen>
</resources>
```

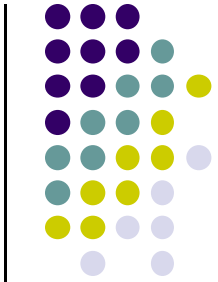
- Can reference “thin” declared above
  - In XML layout files as **@dimen/thin**
  - In Java using **Resources.getDimension(R.dimen.thin)**

Styles

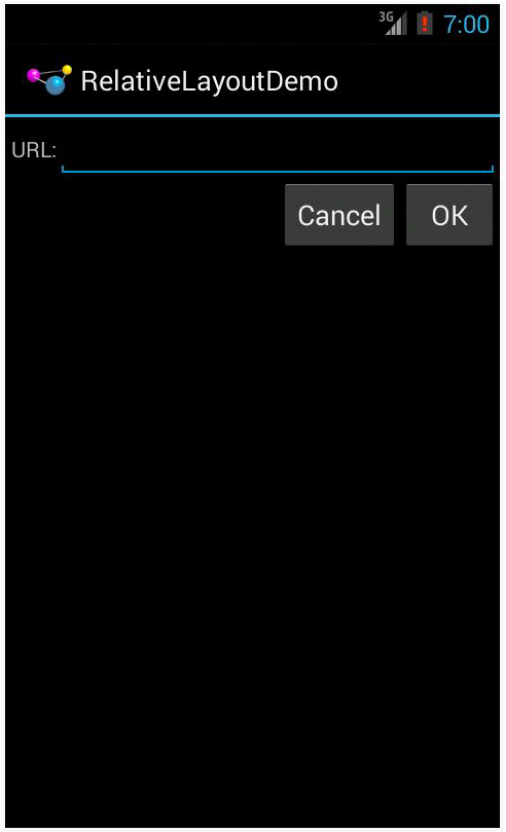


- Styles specify rules for look of Android screen
- Similar to Cascaded Style Sheets (CSS) in HTML
- E.g CSS enables setting look of certain types of tags.
  - E.g. font and size of all <h1> and <h2> elements
- Android widgets have properties
  - E.g. Foreground color = red
- **Styles in Android:** collection of values for properties
- Styles can be specified one by one or themes (e.g. Theme, Theme.holo and Theme.material) can be used

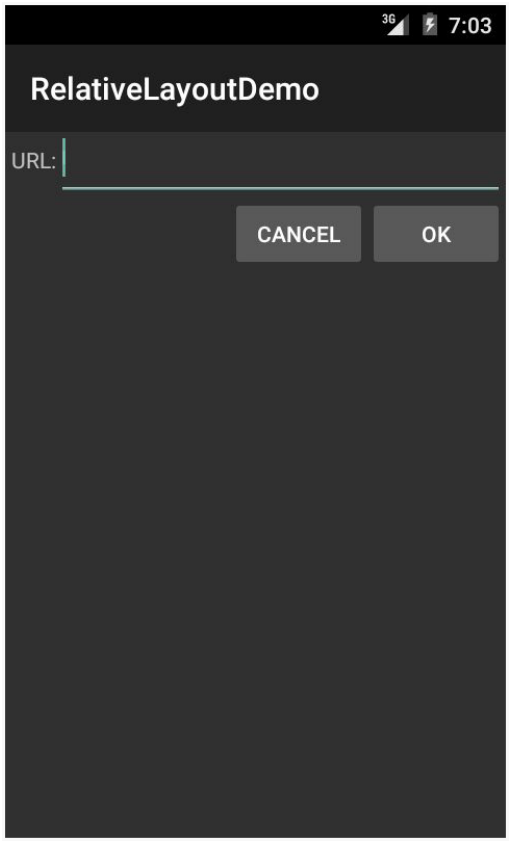
Default Themes



- Android chooses a default theme if you specify none
- Also many stock themes to choose from

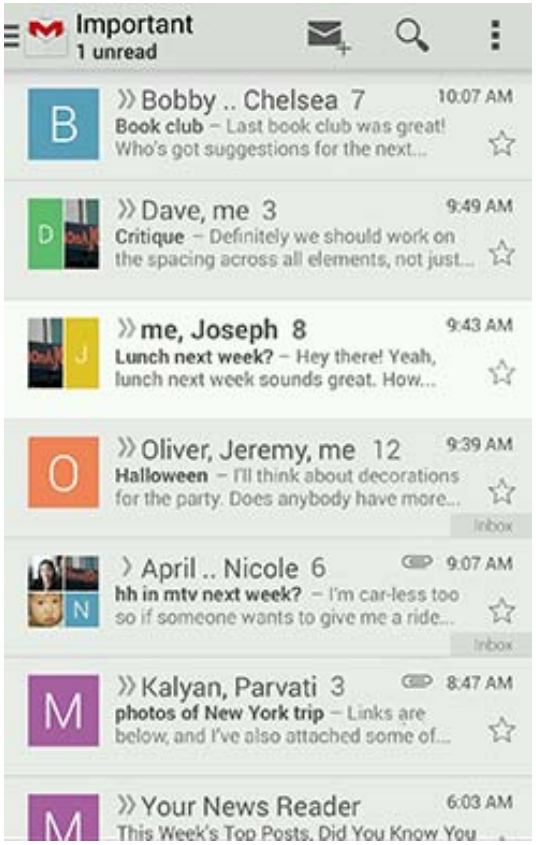


Theme.Holo: default theme in Android 3.0



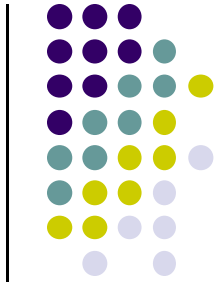
Theme.Material: default theme in Android 5.0

Examples of Themes in Use



GMAIL in Holo Light

Settings screen in Holo Dark

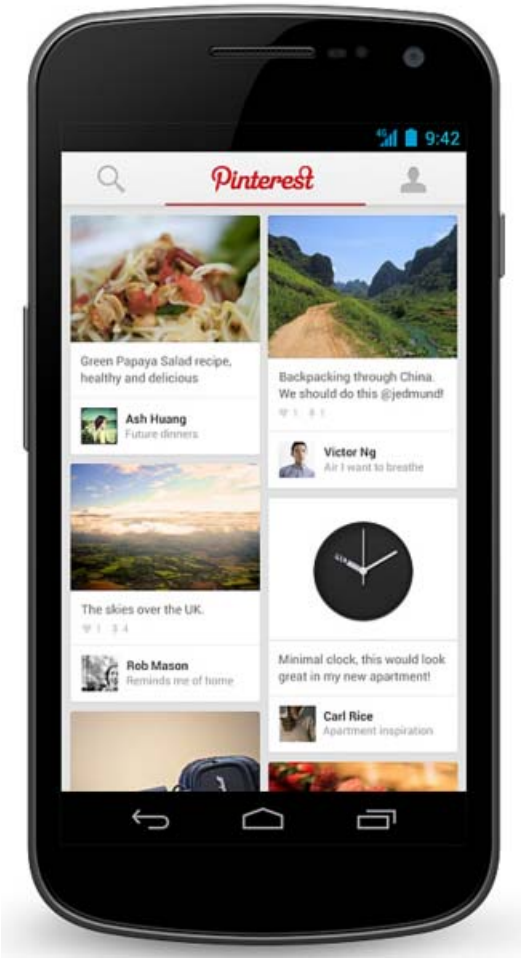


Android UI Design in XML

Recall: Files Hello World Android Project

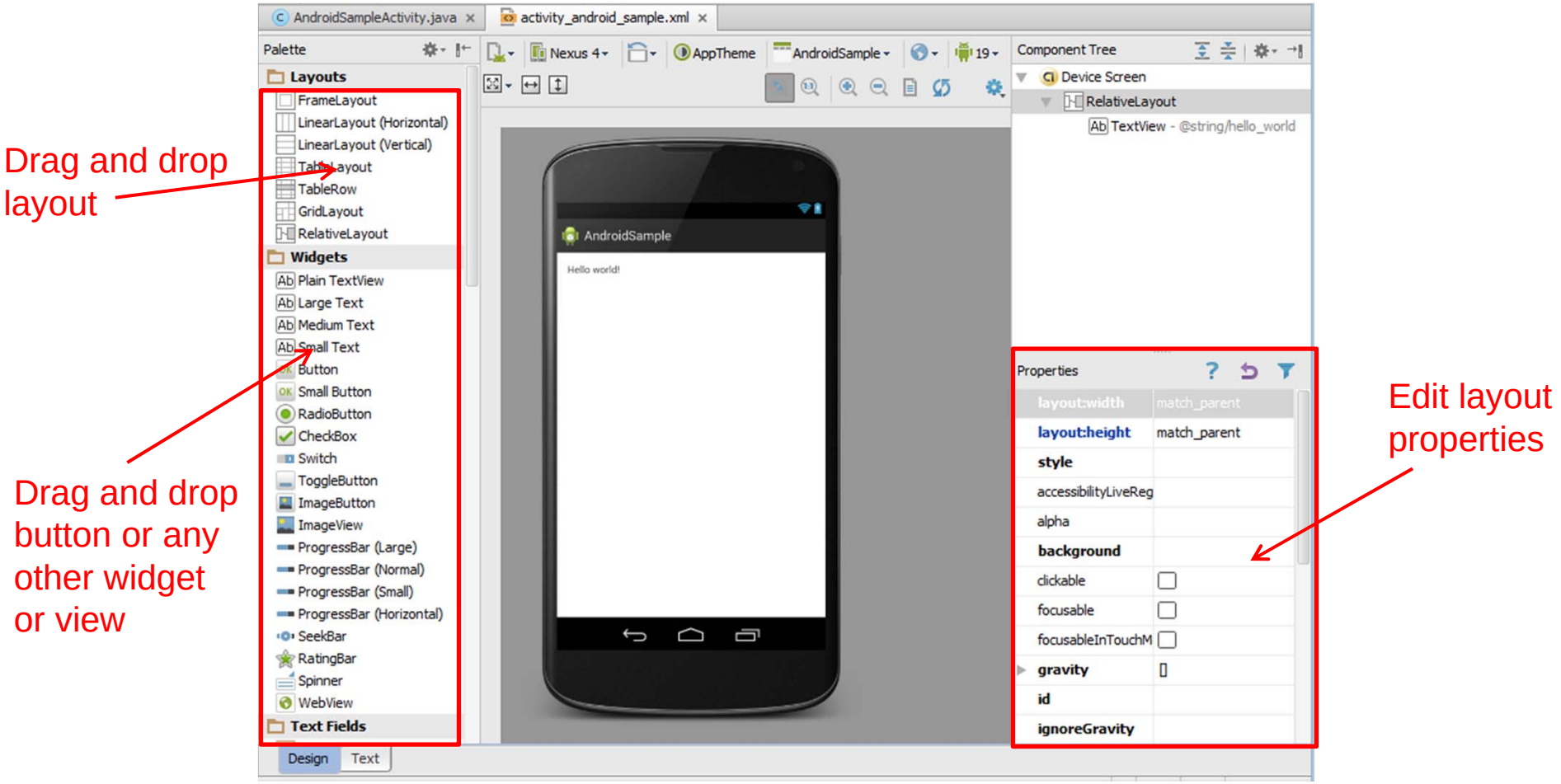
XML file used to design Android UI

- 3 Files:
  - **Activity_my.xml:** XML file specifying screen layout
  - **MainActivity.Java:** Java code to define behavior, actions taken when button clicked (intelligence)
  - **AndroidManifest.xml:**
    - Lists all screens, components of app
    - Analogous to a table of contents for a book
    - E.g. Hello world program has 1 screen, so AndroidManifest.xml has 1 item listed
    - App starts running here (a bit like main( ) in C), launching activity with a tag “LAUNCHER”

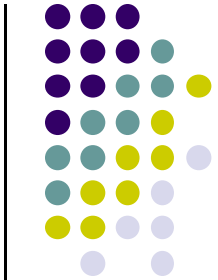


Recall: Edit XML Layouts using Graphical IDE

- Can drag and drop widgets, layouts => XML generated
- Can also edit XML directly
- Can also edit their properties (e.g. height, width, color, etc)



Android UI using XML Layouts



- In XML layout file, we have to choose a layout to use
- Layout? Pattern in which views (boxes) are arranged

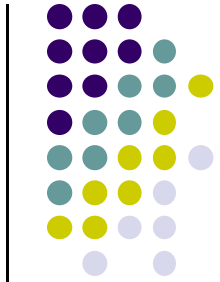
LinearLayout

RelativeLayout

TableLayout

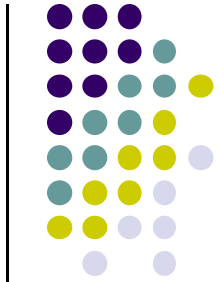
http://developer.android.com/resources/tutorials/views/index.html

Layouts

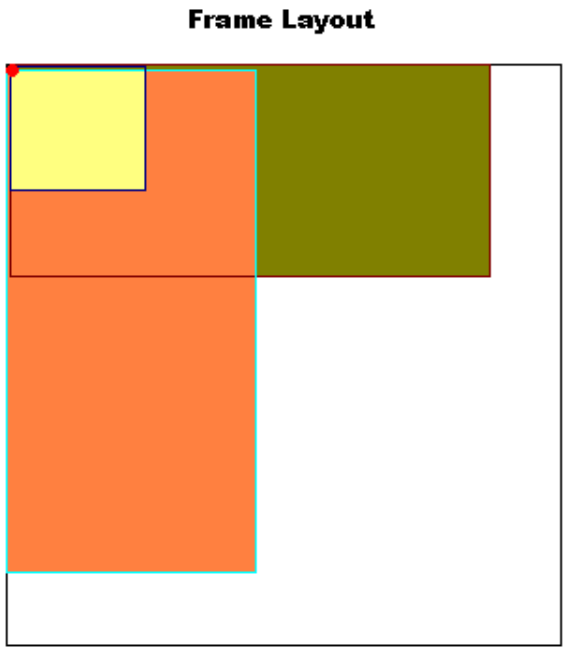


- Layouts can contain widgets, views
- Stored in **res/layout**
- Useful Layouts:
  - FrameLayout,
  - LinearLayout,
  - TableLayout,
  - GridLayout,
  - RelativeLayout,
  - ListView,
  - GridView,
  - ScrollView,
  - DrawerLayout,
  - ViewPager
- More on layouts next

FrameLayout



- FrameLayout
  - simplest type of layout object
  - fill with single object (e.g a picture)
  - child elements pinned to top left corner of screen, cannot be moved
  - adding a new element / child draws over the last one



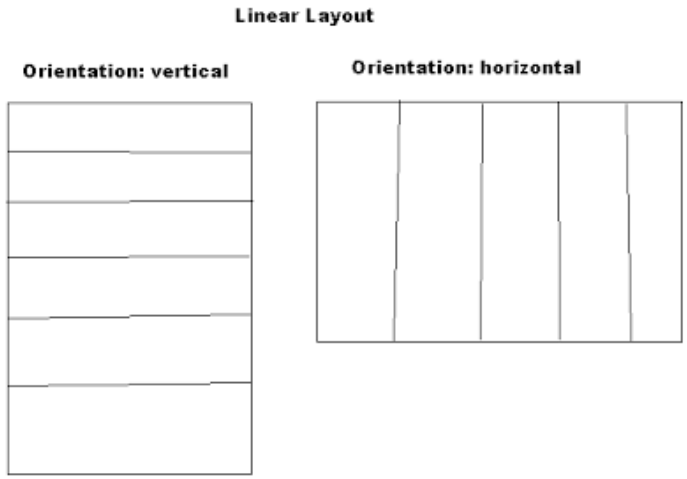
LinearLayout

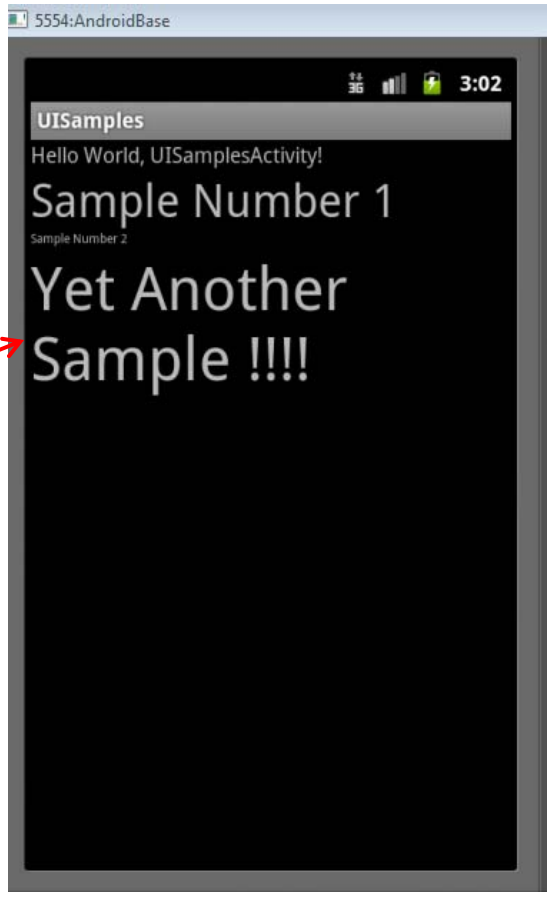
- aligns child elements (e.g. buttons, text boxes, pictures, etc.) in single direction
- Example:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.c
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
        android:background="#ff00ff"
    android:orientation="vertical" >
```

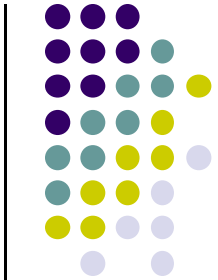
Layout properties

- orientation attribute defines direction (vertical or horizontal): E.g.
  - android:orientation="*vertical*"

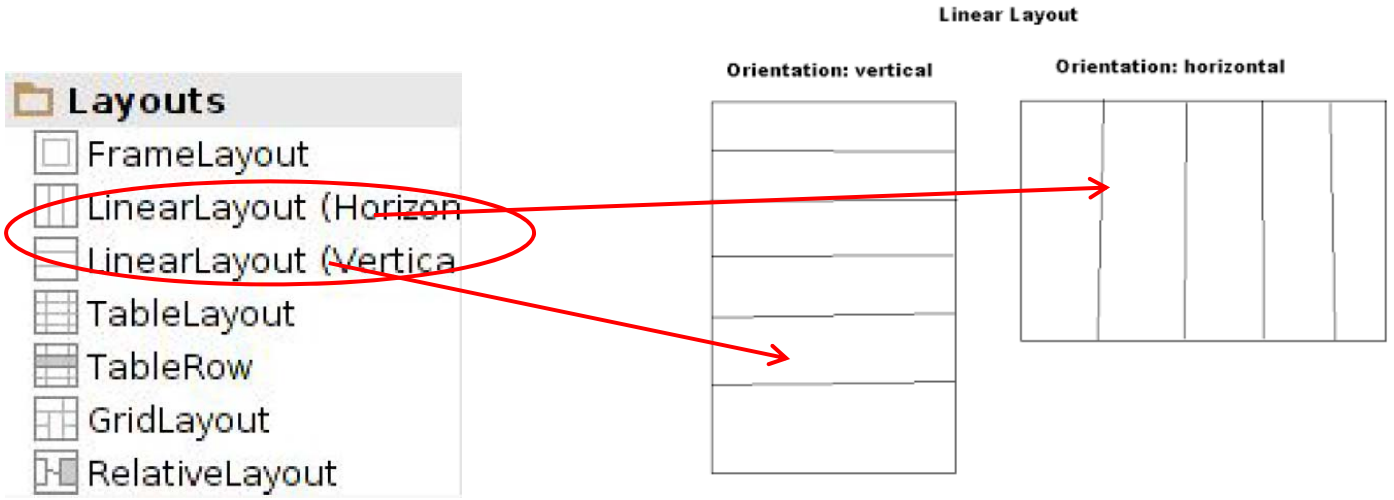




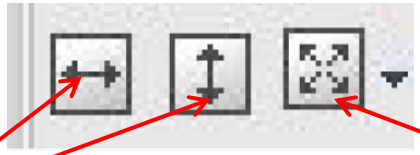
LinearLayout in Android Studio



- LinearLayout can be found in palette of Android Studio Graphical Layout Editor



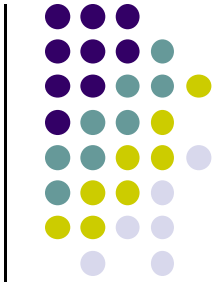
- After selecting LinearLayout, toolbars buttons to set parameters



Toggle width, height between match_parent and wrap_content

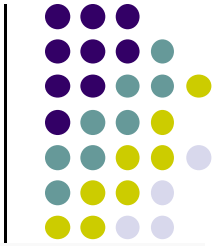
Change gravity of LinearLayout

Attributes



- Layouts have attributes (e.g. width, height, orientation)
- Statements to set attribute values appears in XML file.
- E.g. *android:orientation="vertical"*
- Attributes can be set:
  - In xml file
  - Using IDE (e.g. Android Studio)
  - In Java program
- Lots of attributes!

LinearLayout Attributes

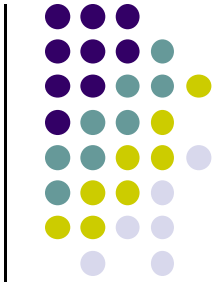


XML Attributes		
Attribute Name	*Related Method*	*Description*
android:baselineAligned	setBaselineAligned(boolean)	When set to false, prevents the layout from aligning its children's baselines.
android:baselineAlignedChildIndex	setBaselineAlignedChildIndex(int)	When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align to (that is, which child TextView).
android:divider	setDividerDrawable(Drawable)	Drawable to use as a vertical divider between buttons.
android:gravity	setGravity(int)	Specifies how to place the content of an object, both on the x- and y-axis, within the object itself.
android:measureWithLargestChild	setMeasureWithLargestChildEnabled(boolean)	When set to true, all children with a weight will be considered having the minimum size of the largest child.
android:orientation	setOrientation(int)	Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.
android:weightSum		Defines the maximum weight sum.

Inherited XML Attributes		[Expand]
▼ From class android.view.ViewGroup		
Attribute Name	*Related Method*	*Description*
android:addStatesFromChildren		Sets whether this ViewGroup's drawable states also include its children's drawable states.
android:alwaysDrawnWithCache		Defines whether the ViewGroup should always draw its children using their drawing cache or not.
android:animateLayoutChanges	setLayoutTransition(LayoutTransition)	Defines whether changes in layout (caused by adding and removing items) should cause a LayoutTransition to run.
android:animationCache		Defines whether layout animations should create a drawing cache for their children.
android:clipChildren	setClipChildren(boolean)	Defines whether a child is limited to draw inside of its bounds or not.
android:clipToPadding	setClipToPadding(boolean)	Defines whether the ViewGroup will clip its drawing surface so as to exclude the padding area.
android:descendantFocusability		Defines the relationship between the ViewGroup and its descendants when looking for a View to take focus.
android:layoutAnimation		Defines the layout animation to use the first time the ViewGroup is laid out.

Can find complete list of attributes, possible values on Android Developer website

Setting Attributes



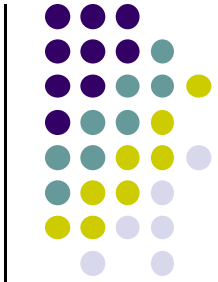
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.c
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
        android:background="#ff00ff"
    android:orientation="vertical" >
```

in layout xml file

```
public class UISamplesActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void change(View v) {
        LinearLayout vg = (LinearLayout)this.findViewById(R.id.main_layout);
        Log.d("UI SAMPLE", vg + "");
        vg.setOrientation(LinearLayout.HORIZONTAL);
    }
}
```

Can also design UI, set attributes in Java program (e.g. ActivityMain.java) (More later)

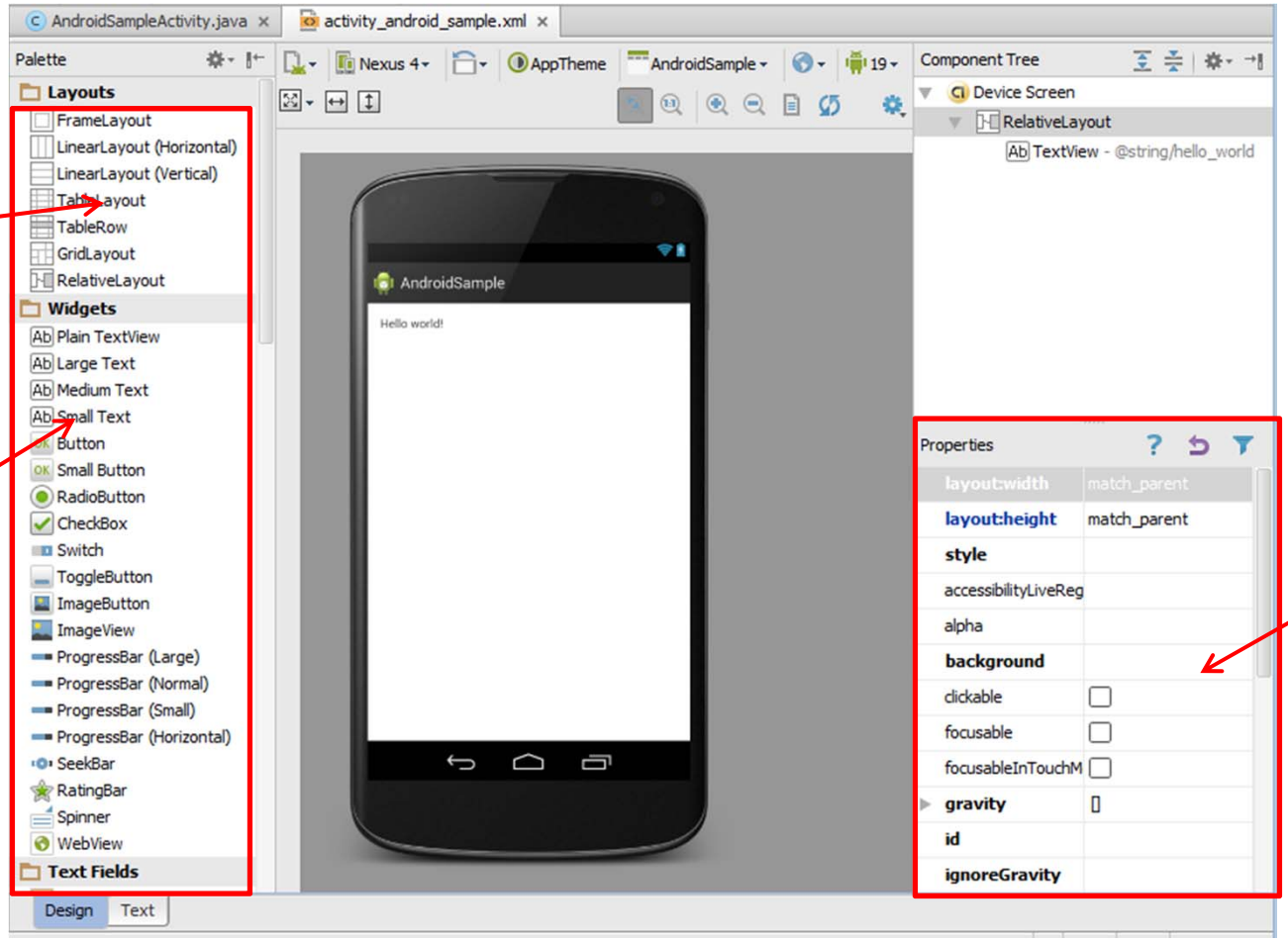
Recall: Edit XML Layouts using Graphical IDE



- Can drag and drop widgets, layouts in Android Studio
- Can also edit their properties (e.g. height, width, color, etc)

Drag and drop layout

Drag and drop button or any other widget or view

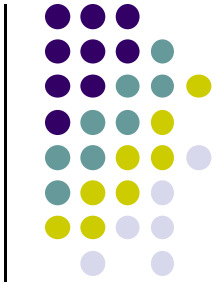


Edit layout attributes

Layout Width and Height Attributes

- **match_parent:** widget as wide/high as its parent
- **wrap_content:** widget as wide/high as its content (e.g. text)
- **fill_parent:** older form of **match_parent**

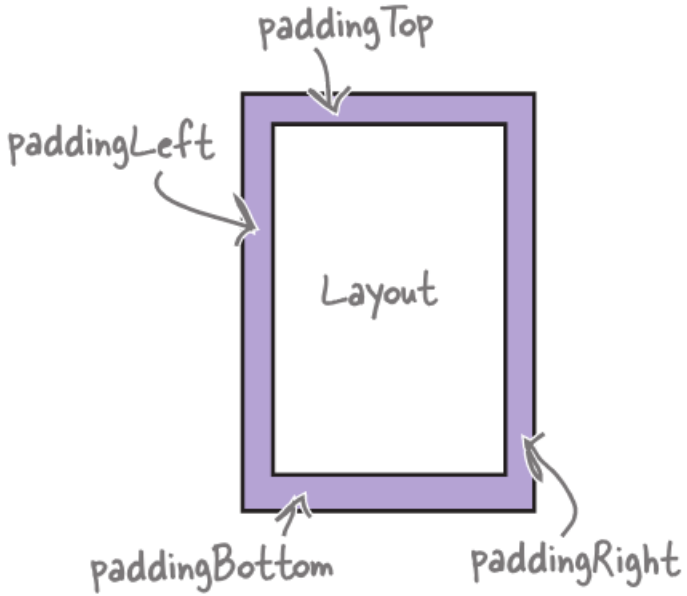
Adding Padding



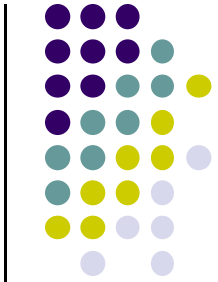
- Paddings sets space between layout sides and its parent

```
<RelativeLayout ...
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp">
    ...
</RelativeLayout>
```

Add padding of 16dp.



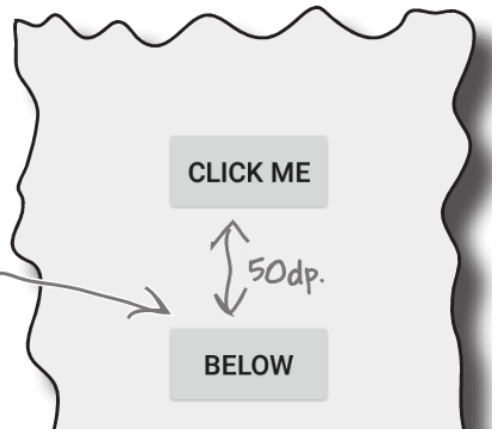
Setting Margins



- Can increase gap (margin) between adjacent views (widgets)
- E.g. To add margin between two buttons, in declaration of bottom button

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button_click_me"
    android:layout_below="@+id/button_click_me"
    android:layout_marginTop="50dp"
    android:text="@string/button_below" />
</RelativeLayout>
```

Adding a margin to the top of the bottom button adds extra space between the two views.

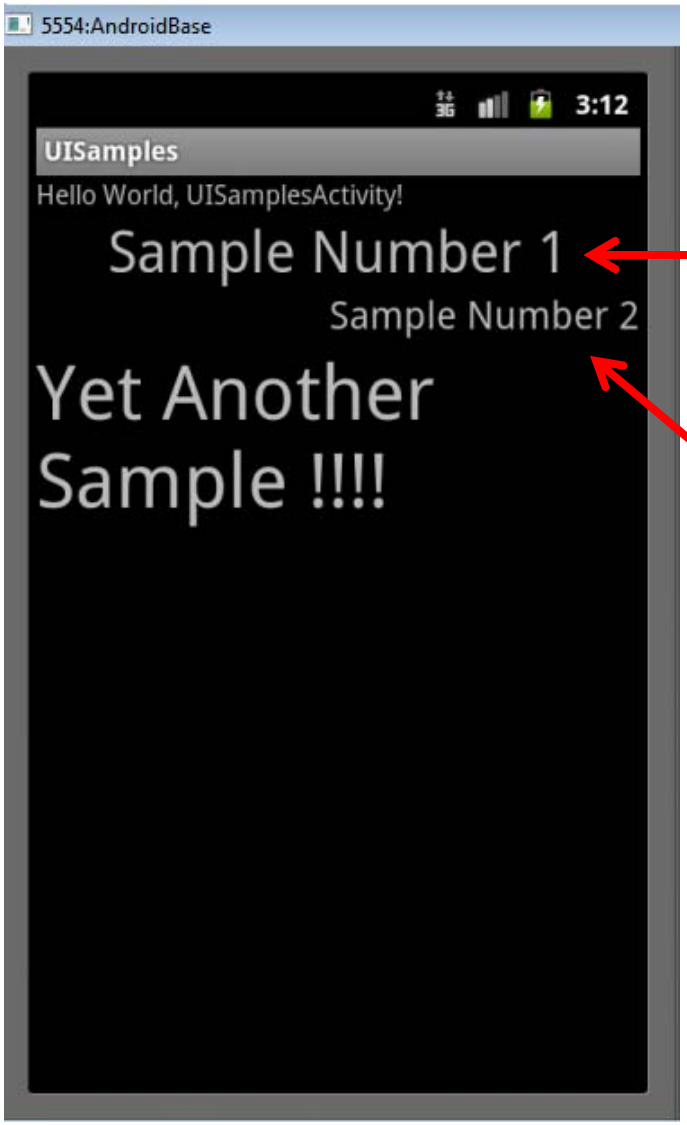


- Other options

android:layout_marginLeft

android:layout_marginRight

Gravity Attribute



- By default, linearlayout left- and top-aligned
- Gravity attribute can change position of :
  - Widget within Linearlayout
  - Contents of widgets (e.g. android:gravity = “right”)

Weight

- layout_weight attribute
  - Specifies "importance" of a view (i.e. button, text, etc)
  - default = 0
  - Larger weights (layout_weight > 0) takes up more of parent space

Another Weight Example

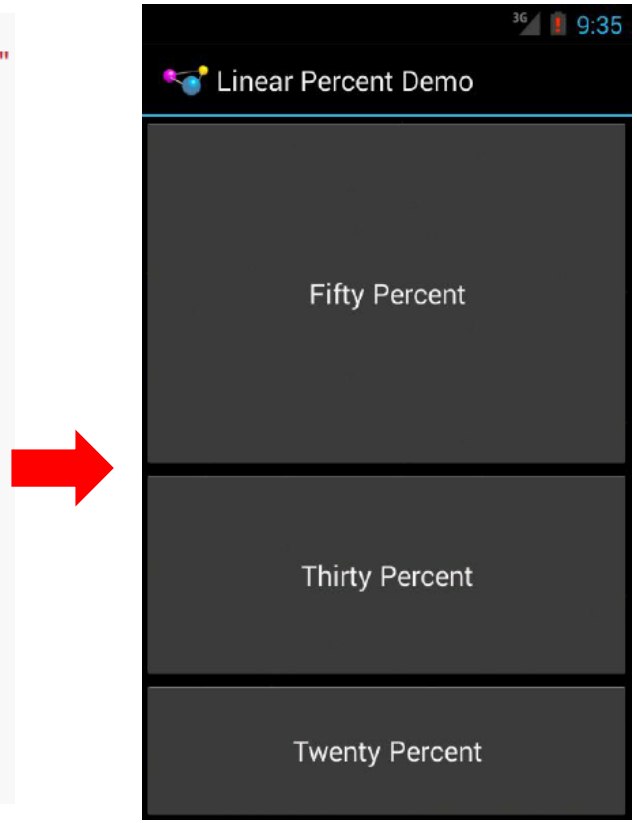
button and bottom edit text weight of 2

button weight 1 and bottom edit text weight of 2

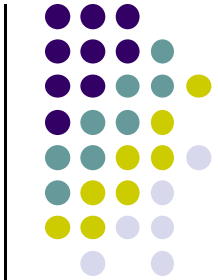
Linear Layout

- Alternatively, set
  - width, height = 0 then
  - weight = percent of height/width you want element to cover

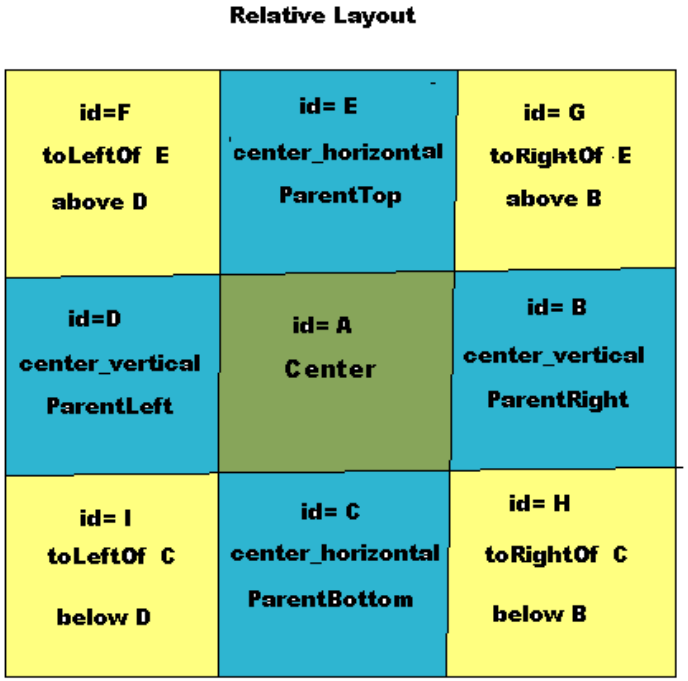
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">

  <Button
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="50"
    android:text="@string/fifty_percent"/>

  <Button
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="30"
    android:text="@string/thirty_percent"/>

  <Button
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="20"
    android:text="@string/twenty_percent"/>

</LinearLayout>
```

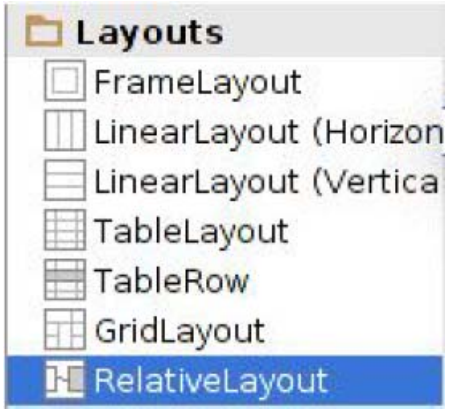


RelativeLayout



- First element listed is placed in "center"
- Positions of children specified relative to parent or to each other.
  - E.g. **android:layout_toRightOf = “true”:** widget should be placed to the right of widget referenced in the property
  - **android:layout_alignParentBottom = “true”:** align widget’s bottom with layout’s bottom





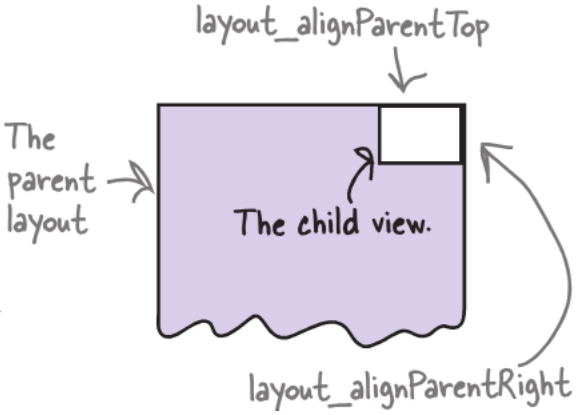
RelativeLayout available In Android Studio palette

Positioning Views Relative to Parent Layout

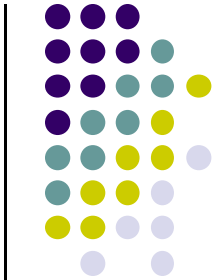
- Can position a view (e.g. button, TextView) relative to its parent
- Example: A button in a Relative Layout

```
<RelativeLayout ... >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click_me"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
</RelativeLayout>
```

The layout contains the button, so the layout is the button's parent.

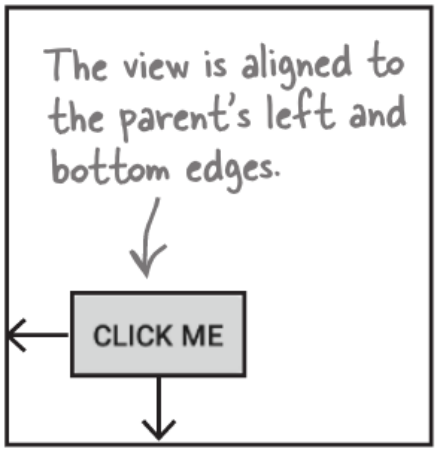


Examples: Positioning a Button Relative to Parent Layout



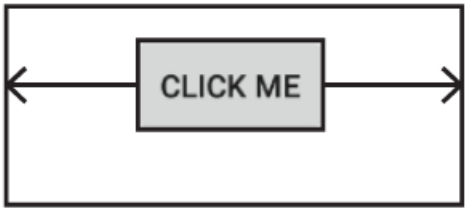
- Align to parent bottom and left

android:
layout_alignParentBottom



android:
layout_alignParentLeft

android:
layout_centerHorizontal



See Head First Android Development page 169 for more examples

Positioning Views Relative to Other Views

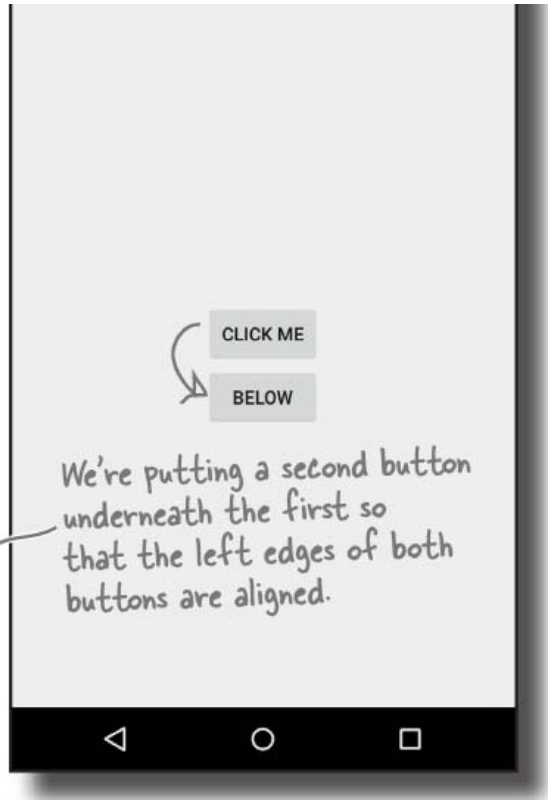
- The anchor view has to be assigned an ID using **android:id**
- **E.g.** Relative layout with 2 buttons (1 centered in layout middle, second button underneath first button)

Assign anchor button an ID

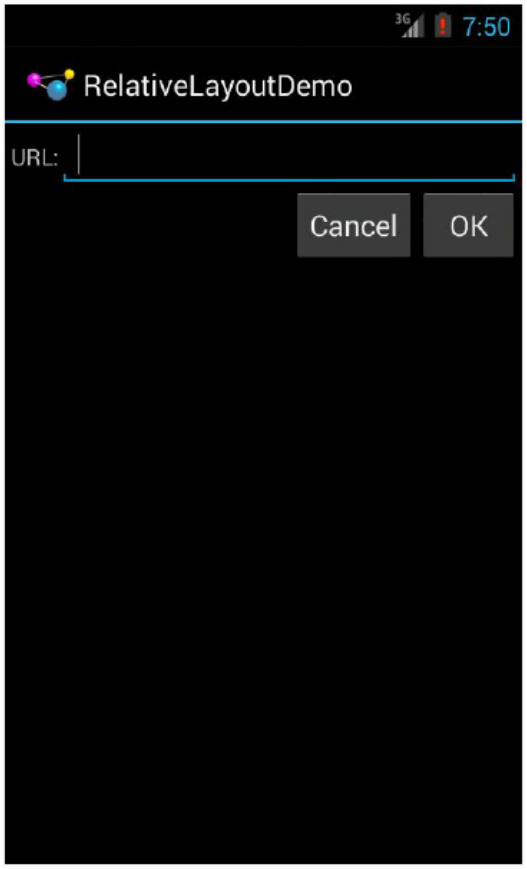
Align second button to first button's left and below

```
<RelativeLayout ... >
    <Button
        android:id="@+id/button_click_me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="@string/click_me" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button_click_me"
        android:layout_below="@+id/button_click_me"
        android:text="@string/new_button_text" />
</RelativeLayout>
```

We're using this button as an anchor for the second one, so it needs an ID.



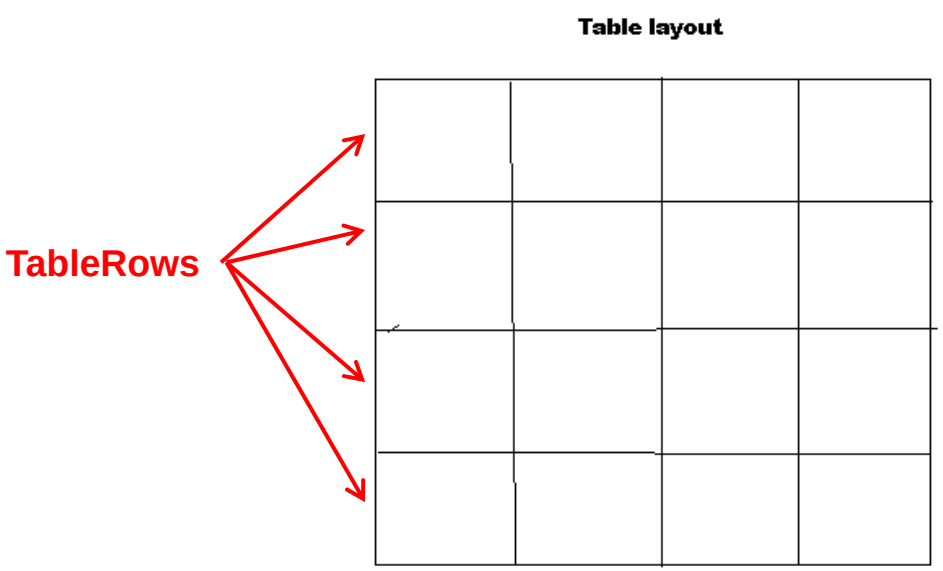
```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content">

  <TextView
    android:id="@+id/label"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/entry"
    android:layout_alignParentLeft="true"
    android:layout_marginLeft="4dip"
    android:text="@string/url"/>

  <EditText
    android:id="@id/entry"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@id/label"
    android:inputType="text"/>

  <Button
    android:id="@+id/ok"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@id/entry"
    android:layout_below="@id/entry"
    android:text="@string/ok"/>

  <Button
    android:id="@+id/cancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@id/ok"
    android:layout_toLeftOf="@id/ok"
    android:text="@string/cancel"/>

</RelativeLayout>
```

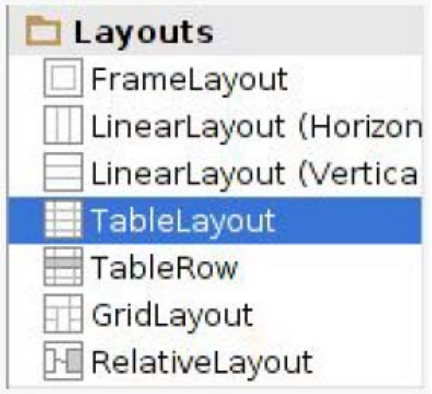


RelativeLayout XML Example

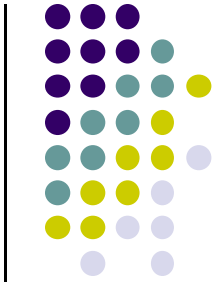
Table Layout

- Specify number of rows and columns of views.
- Available in Android Studio palette

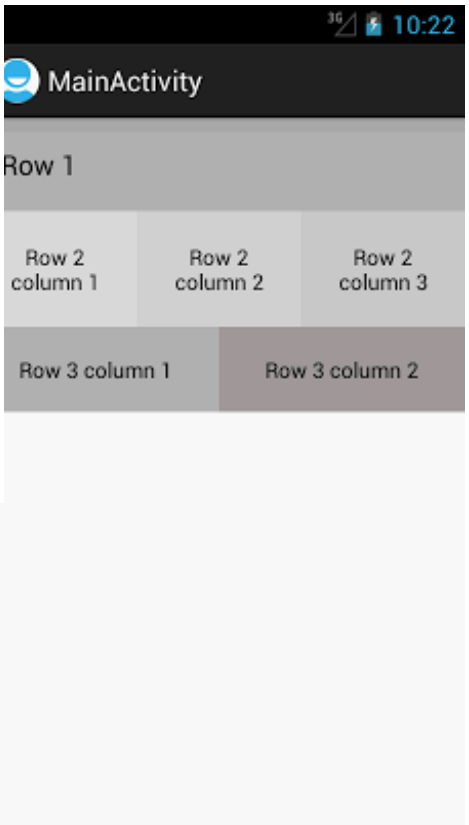




TableLayout Example

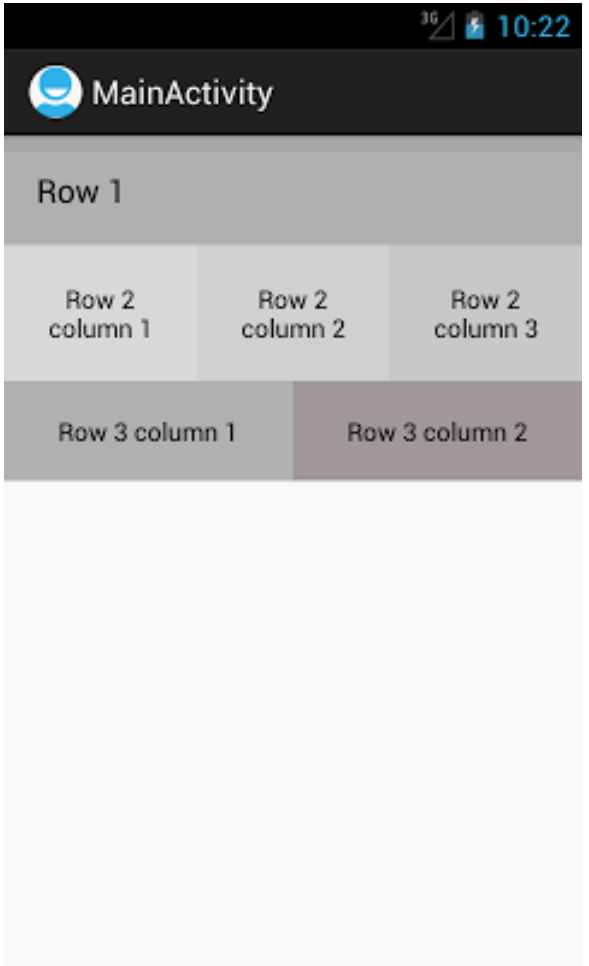


```
<TableLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:shrinkColumns="*"
        android:stretchColumns="*"
        android:background="#ffffff">

            <!-- Row 1 with single column -->

            <TableRow
               android:layout_height="wrap_content"
               android:layout_width="fill_parent"
               android:gravity="center_horizontal">

               <TextView
                  android:layout_width="match_parent"
                  android:layout_height="wrap_content"
                  android:textSize="18dp"
                  android:text="Row 1"
                  android:layout_span="3"
                  android:padding="18dip"
                  android:background="#b0b0b0"
                  android:textColor="#000"/>

            </TableRow>
```



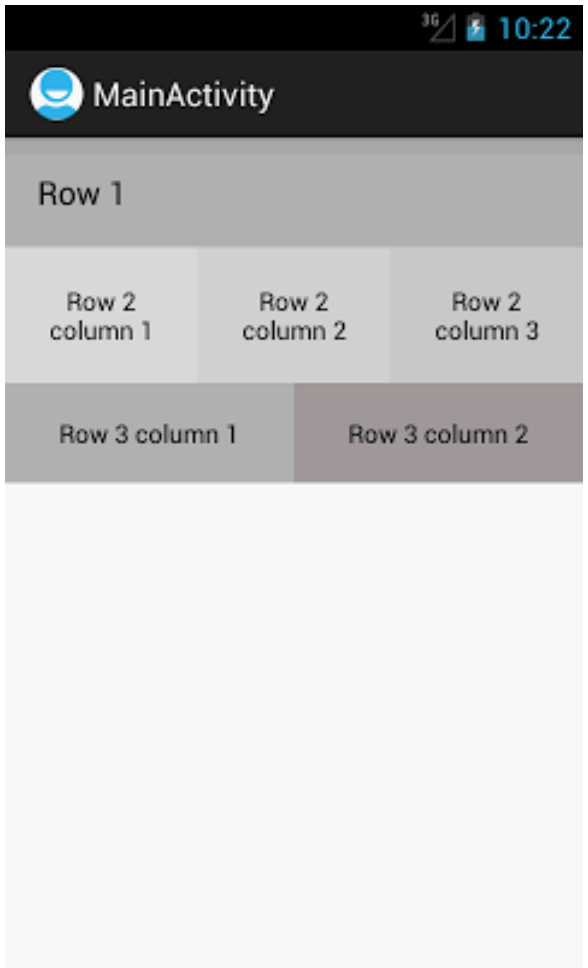
TableLayout Example

```
<!-- Row 2 with 3 columns -->

<TableRow
    android:id="@+id/tableRow1"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">

    <TextView
        android:id="@+id/TextView04"
        android:text="Row 2 column 1"
        android:layout_weight="1"
        android:background="#dcdcdc"
        android:textColor="#000000"
        android:padding="20dip"
        android:gravity="center"/>
    <TextView
        android:id="@+id/TextView04"
        android:text="Row 2 column 2"
        android:layout_weight="1"
        android:background="#d3d3d3"
        android:textColor="#000000"
        android:padding="20dip"
        android:gravity="center"/>
    <TextView
        android:id="@+id/TextView04"
        android:text="Row 2 column 3"
        android:layout_weight="1"
        android:background="#cac9c9"
        android:textColor="#000000"
        android:padding="20dip"
        android:gravity="center"/>
</TableRow>
```

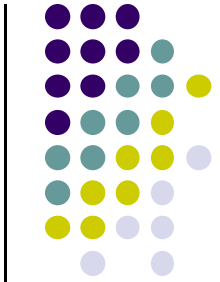


TableLayout Example

```
<!-- Row 3 with 2 columns -->

<TableRow
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:gravity="center_horizontal">

    <TextView
        android:id="@+id/TextView04"
        android:text="Row 3 column 1"
        android:layout_weight="1"
        android:background="#b0b0b0"
        android:textColor="#000000"
        android:padding="18dip"
        android:gravity="center"/>

    <TextView
        android:id="@+id/TextView04"
        android:text="Row 3 column 2"
        android:layout_weight="1"
        android:background="#a09f9f"
        android:textColor="#000000"
        android:padding="18dip"
        android:gravity="center"/>
</TableRow>

</TableLayout>
```



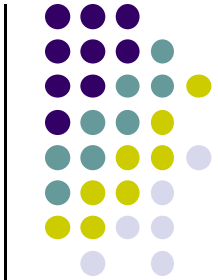
GridLayout



- Added in Android 4.0 (2011)
- In TableLayout, Rows can span multiple columns only
- In GridLayout, child views/controls can span multiple rows **AND** columns
  - different from TableLayout
- Gives greater design flexibility

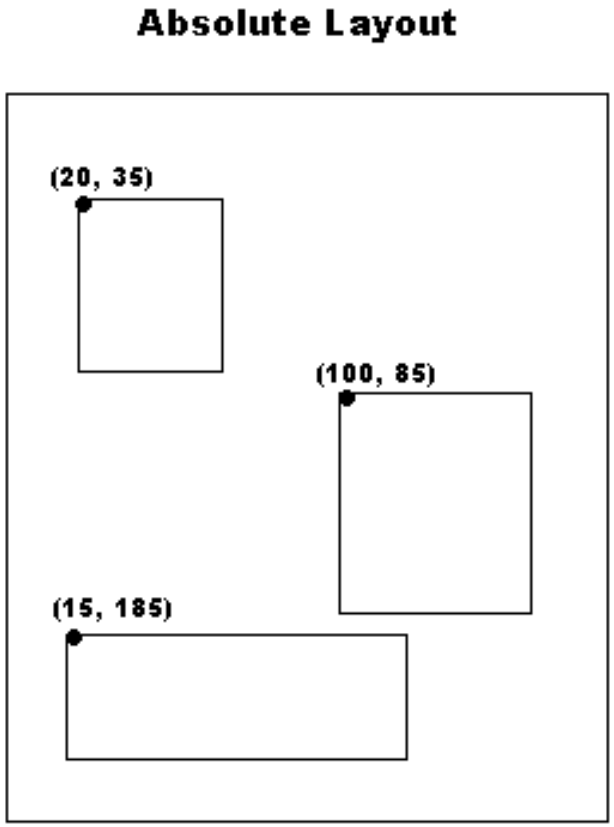
- See section “GridLayout Displays Views in a Grid” in Head First Android Development (pg 189)

Absolute Layout



- Allows specificification of exact locations (x/y coordinates) of its children.

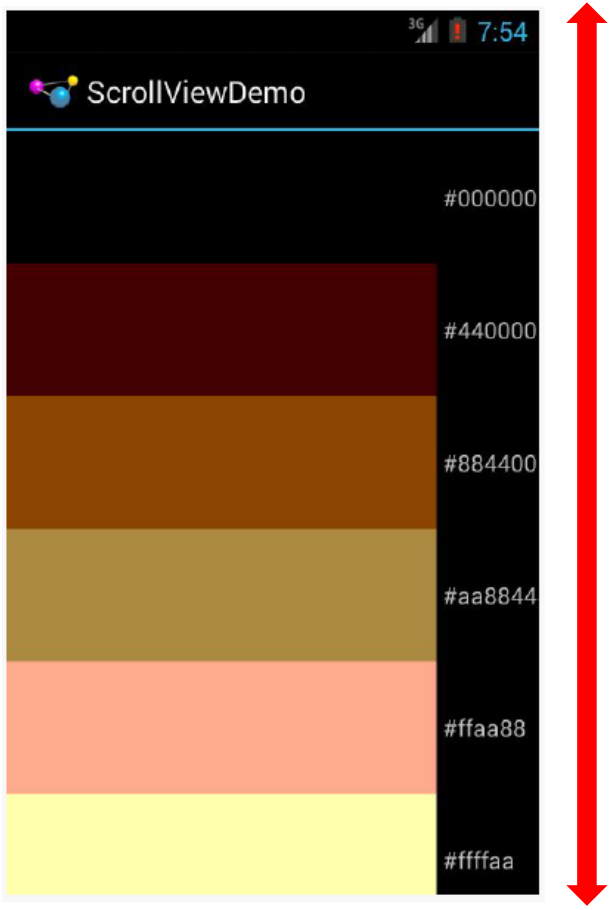
- Less flexible, harder to maintain



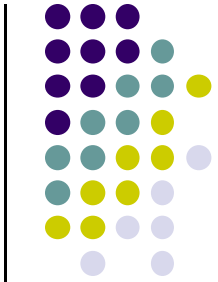
Scrolling

- Phone screens are small, scrolling content helps
- ListView supports vertical scrolling
- Other views for Scrolling:
  - **ScrollView** for vertical scrolling
  - **HorizontalScrollView**
- Examples:
  - scroll through large image
  - Linear Layout with lots of elements
- Rules:
  - Only one direct child View
  - Child could have many children of its own

```
<ScrollView
    ...>
    <LinearLayout>
       ....
       <!-- you can have as many Views in here as you want -->
    </LinearLayout>
</ScrollView>
```



Other Layouts - Tabbed Layouts

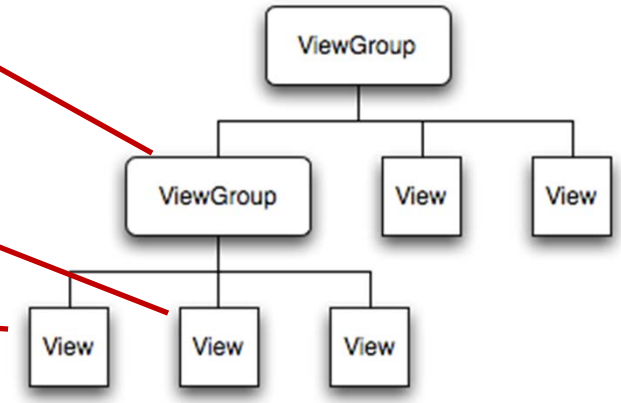


Android Views, Widgets and ViewGroups

Views and ViewGroups

- A view (e.g. buttons, text fieds) is basic UI building block
- View occupies rectangular area on screen
- ViewGroup (e.g. a layout) contains multiple Views

```
<?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" >

    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

    <Button
        android:id="@+id/hello_button"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Press Me" />

</LinearLayout>
```



Layouts (e.g. linear layout, Relative layout)

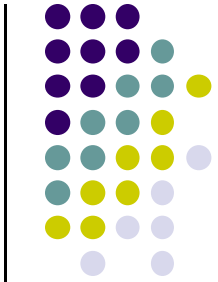
TextViews (labels), ImageViews, Controls such as buttons, etc.

Tree from: http://developer.android.com/guide/topics/ui/index.html

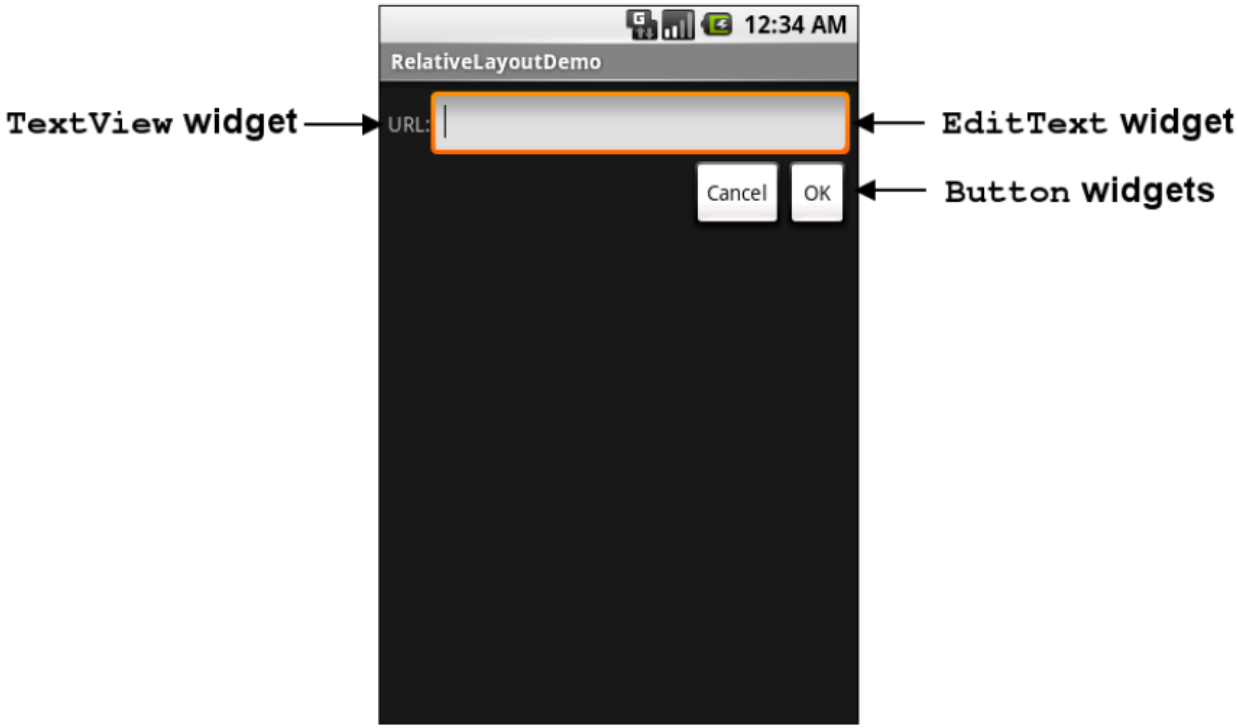
Widgets

- Widgets are visual building blocks used to compose Android screens (Activities)
- Need to specify size, margins and padding of widgets

Widgets

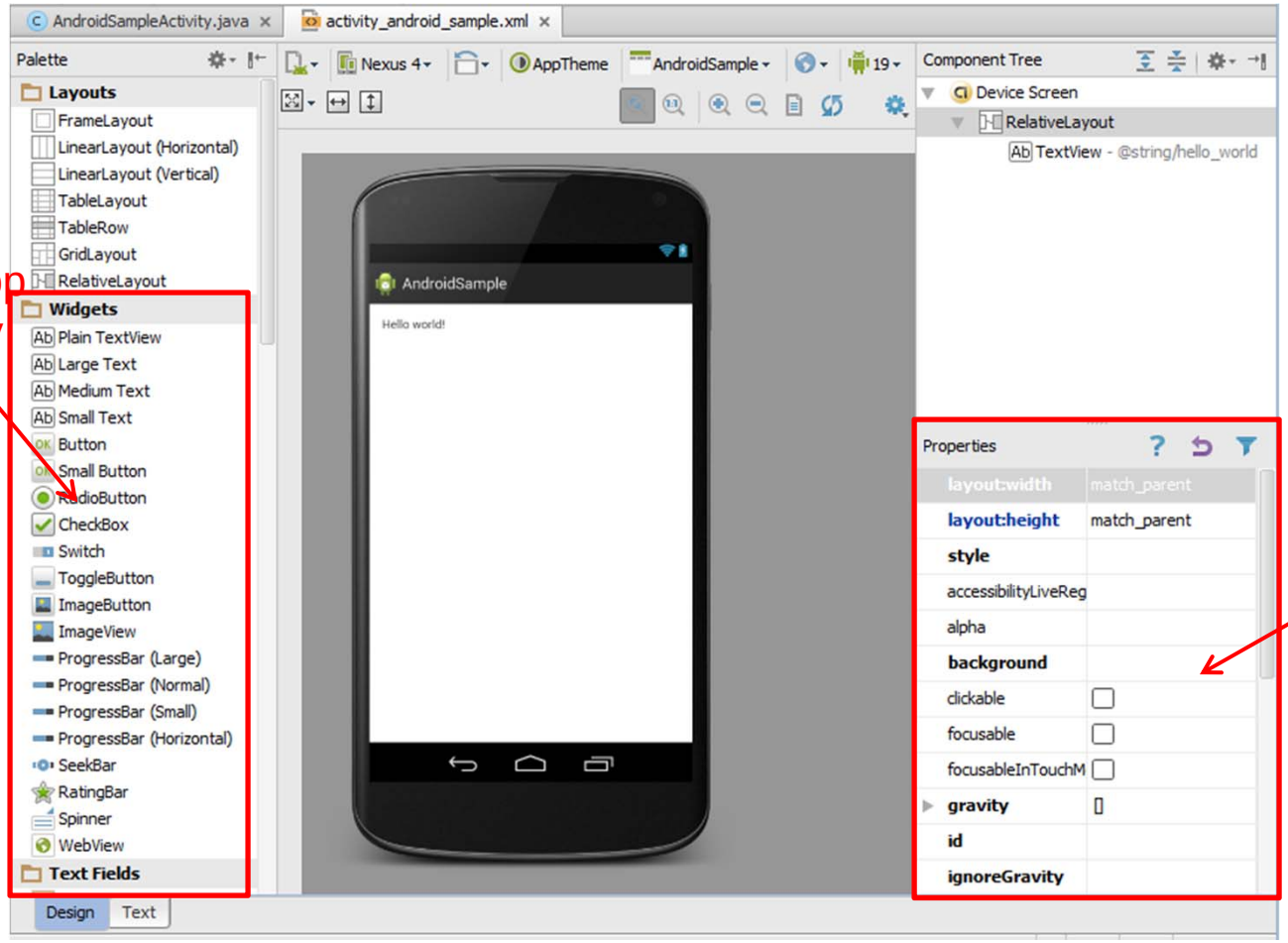


- Most Android UI developed using widgets (fields, lists, text boxes, buttons, etc)
- Example: Screen showing 3 widgets



Adding Widgets in Android Studio

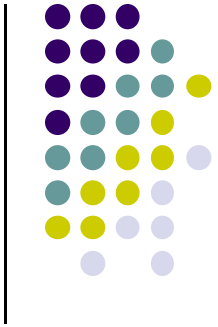
- Can drag and drop widgets, layouts in Android Studio
- Can also edit their properties (e.g. height, width, color, etc)



Drag and drop button or any other widget or view

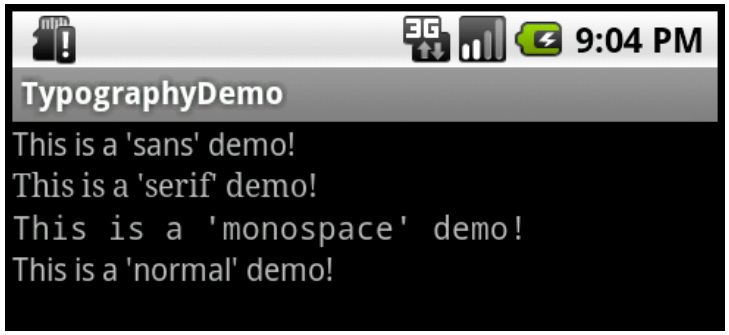
Edit widget properties

TextView



- Text in a rectangle, a simple label
- display information, not for interaction
- **Common attributes:**
  - typeface (android:typeface e.g monospace), bold, italic, (android:textStyle), text size, text color (android:textColor e.g. #FF0000 for read), width, height, padding, visibility, background color
  - set number of lines of text that are visible
    - android:lines="2"
  - Links to email address, url, phone number,
    - web, email, phone, map, etc

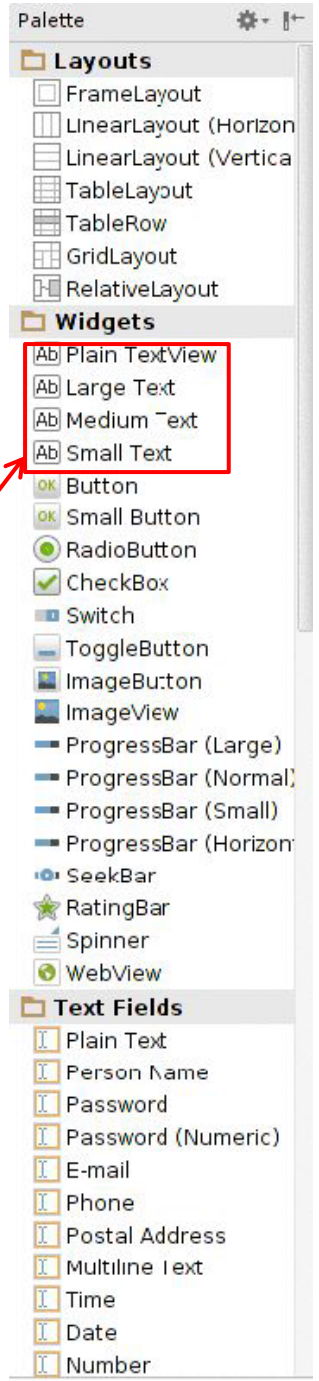
```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="This is a 'sans' demo!"
    android:typeface="sans"
/>
```



TextView

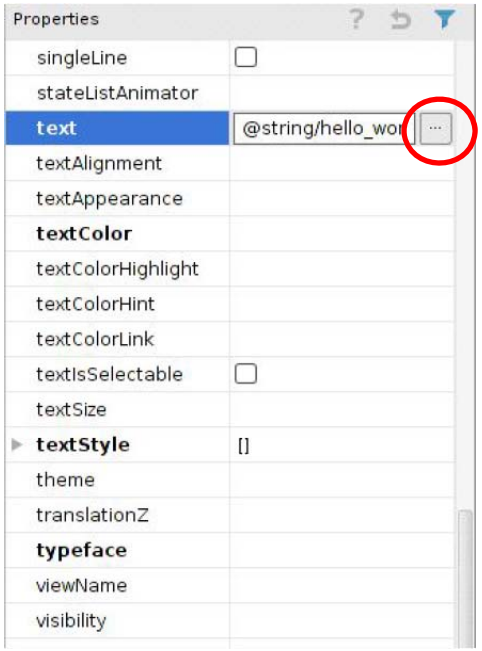
- TextView widget is available in widgets palette in Android Studio Layout editor

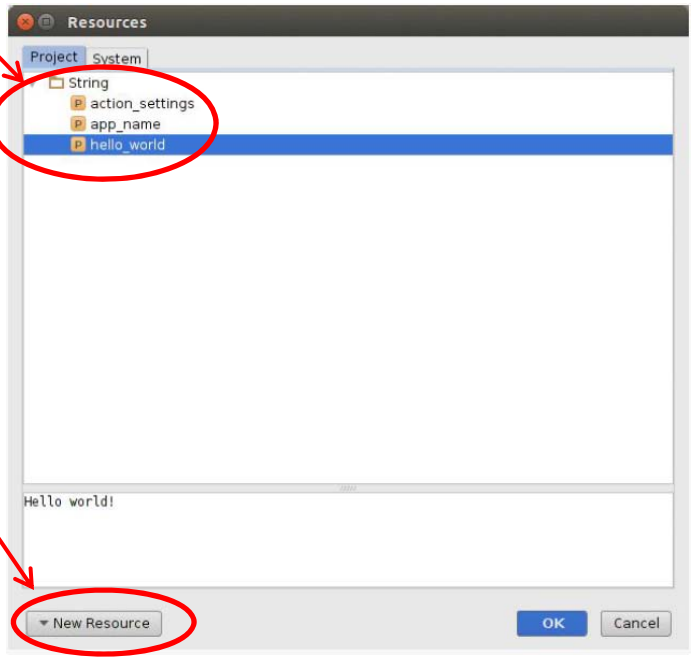
- **Plain TextView**, **Large text, Medium text** and **Small text** are all TextView widgets



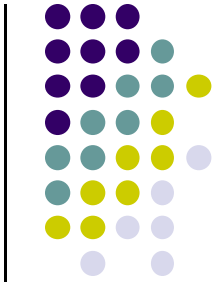
Setting Text Properties

- Can edit text properties
- Can
  - Type in literal string
  - Pick previously declared a string (e.g. in strings.xml)
  - Declare new string by clicking on “New Resource”

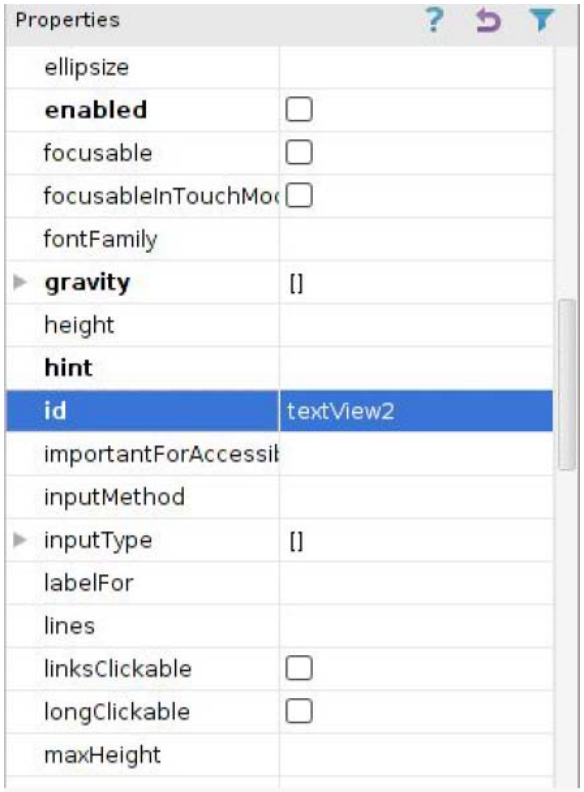




Widget ID

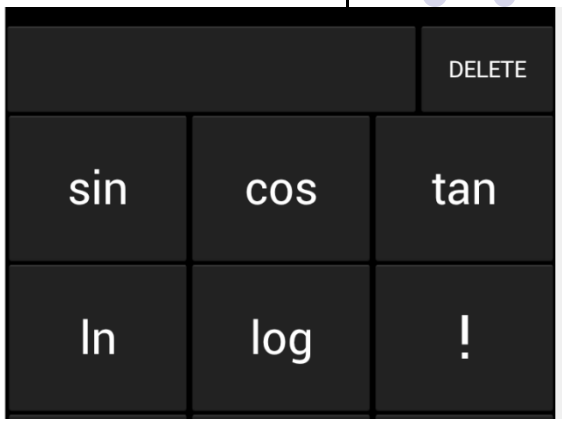


- Every widget has ID whose value is stored in **android:id** attribute
- To manipulate this widget or set its attributes in Java code, need to reference it using its ID
- More on this later
- Naming convention
  - First time use: @+id/xyx_name
  - Subsequent use: @id/xyz_name

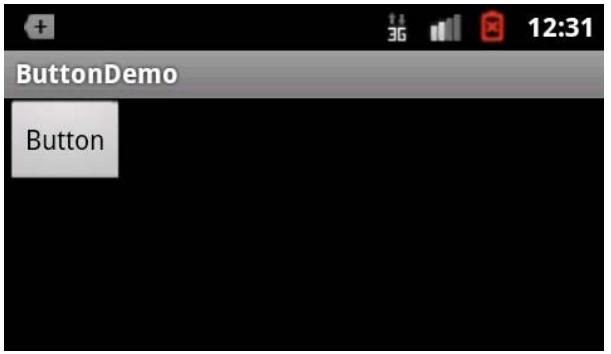


Button Widget

- Text or icon or both on View (Button)
- E.g. “Click Here”
- Appearance of buttons can be customized
  http://developer.android.com/guide/topics/ui/controls/button.html#CustomBackground
- Declared as subclass of TextView so similar attributes



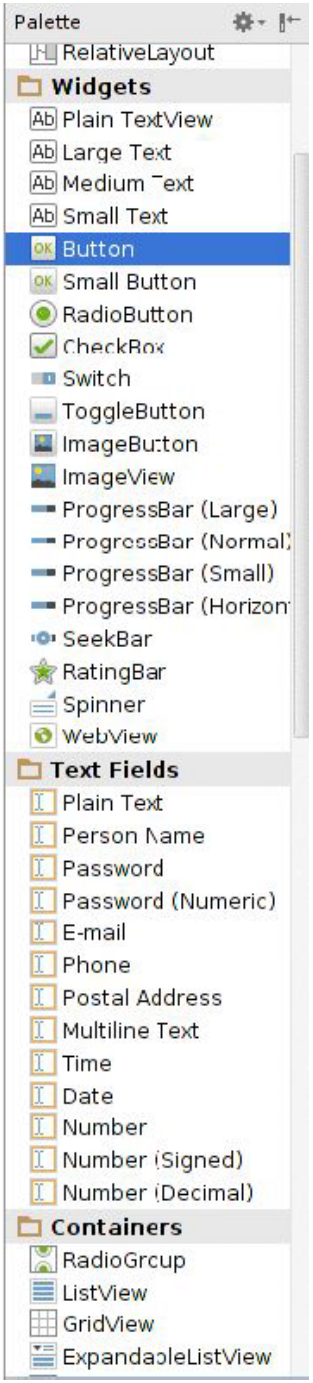

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button"/>

</LinearLayout>
```



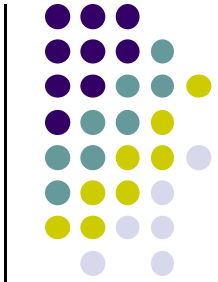
Button in Android Studio

- **Button** widget available in palette of Android Studio graphical layout editor

- Can drag and drop button, edit attributes as with TextView

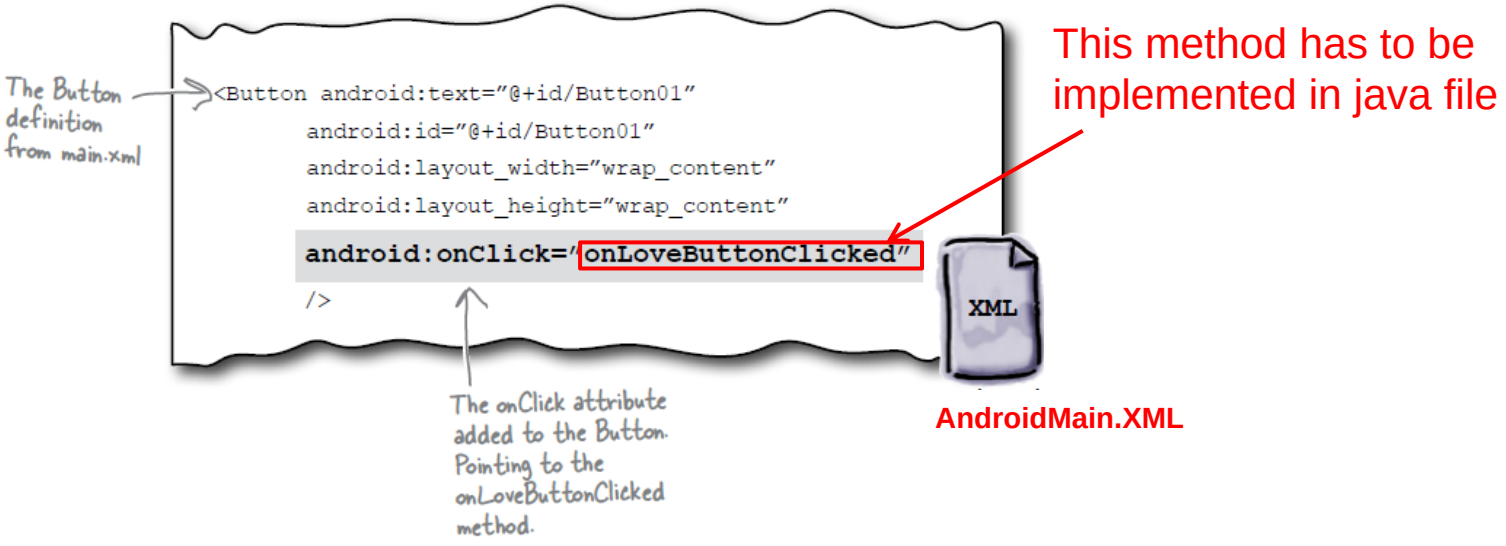


Example: Make Button Responding to Clicks

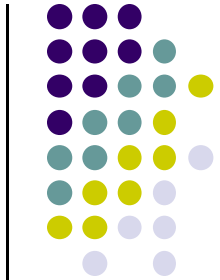


- **Task:** Display some text when user clicks a button

- In declaration of the button, add property “onClick”, give name of method to call onClick



Responding to Button Clicks



- May want Button press to trigger some action
- How?

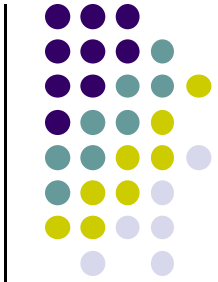
1. In XML file (e.g. Activity_my.xml), set android:onClick attribute to specify method to be invoked

```
<Button
  android:onClick="someMethod"
  ...
/>
```

2. In Java file (e.g. MainActivity.java) declare method/handler to take desired action

```
public void someMethod(View theButton) {
  // do something useful here
}
```

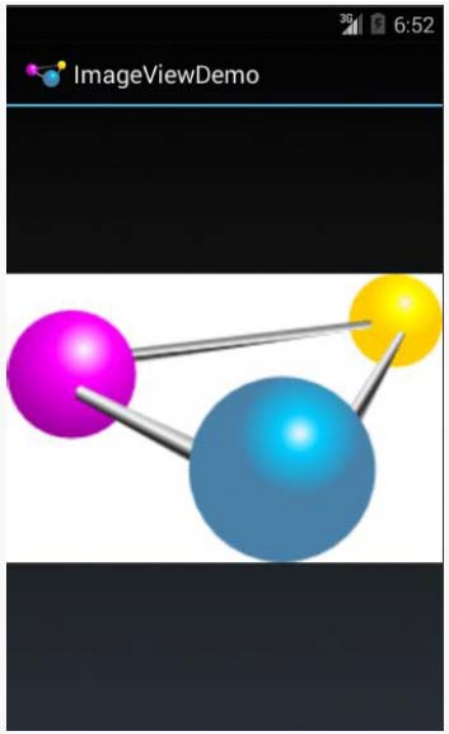
Embedding Images: ImageView and ImageButton



- **ImageView** and **ImageButton:** Image-based based analogs of TextView and Button
  - **ImageView:** display image
  - **ImageButton:** Clickable image
- Use attribute **android:src** to specify image source in **drawable** folder (e.g. **@drawable/icon**)

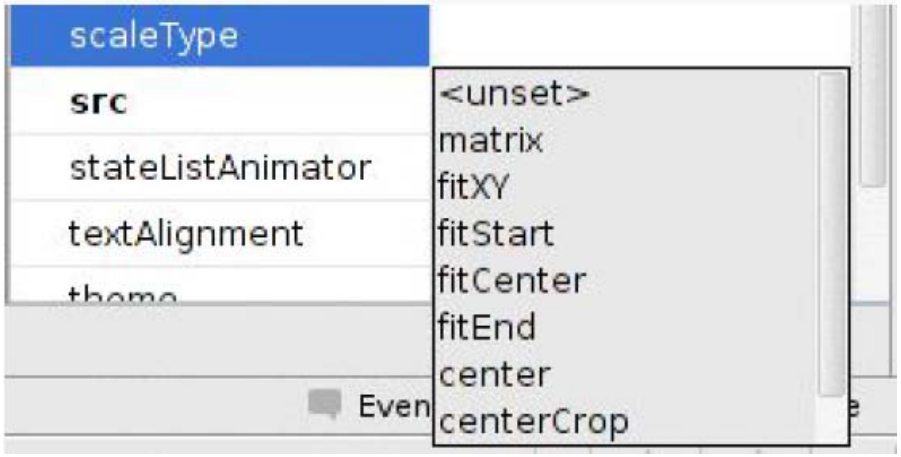
```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/icon"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:adjustViewBounds="true"
  android:src="@drawable/molecule"/>
```

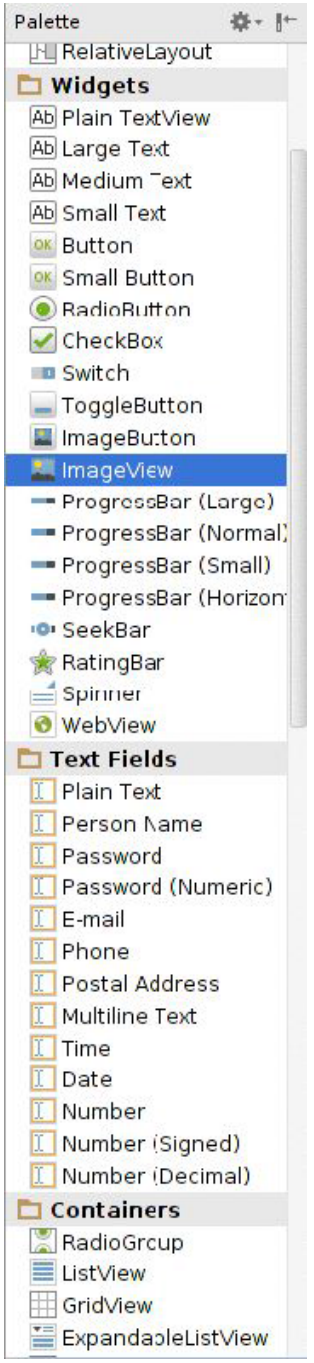
File molecule.png in drawable/ folder



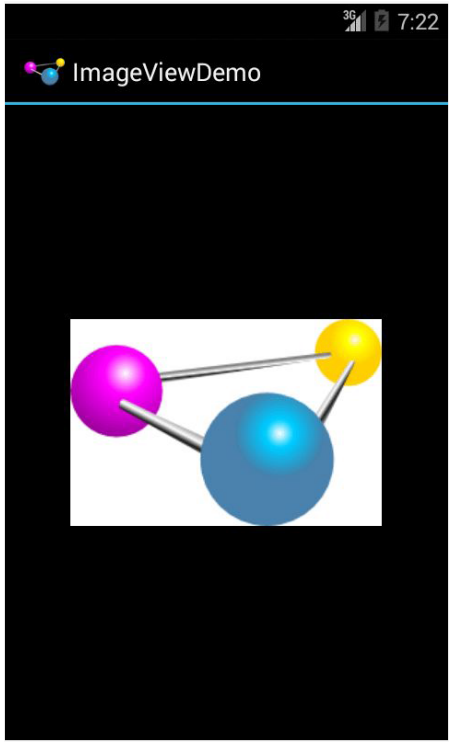
ImageView in Widgets Palette

- Can drag and drop ImageView from Widgets Palette
- Can also use menus to specify:
  - **src:** to choose image to be displayed
  - **scaleType:** to choose how image should be scaled

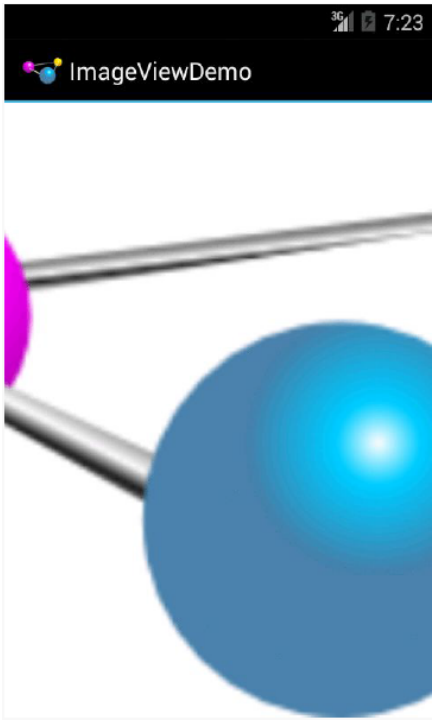




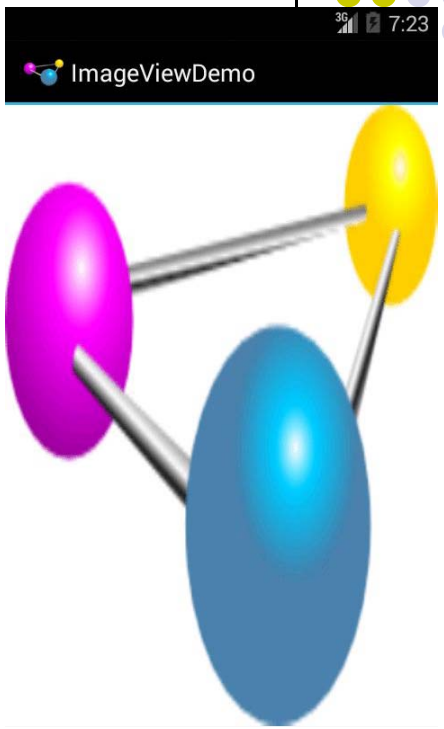
Options for Scaling Images (scaleType)



“center” centers image but does not scale it



“centerCrop” centers images, scales it so that shortest dimension fills available space, and crops longer dimension

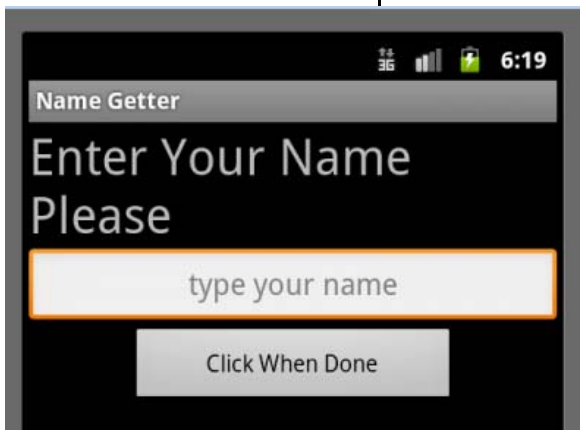


“fitXY” scales image to fit ImageView, ignoring aspect ratio

EditText Widget

- UI Component used for user input
- long press brings up context menu
- Example:

```
<EditText
    android:id="@+id/edittext"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:inputType="textPersonName"
    android:hint="type your name" />
```

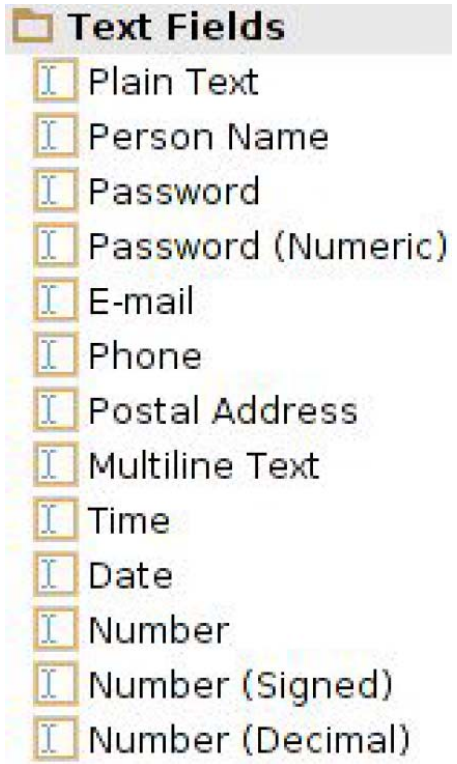


- Text fields can have different input types such as number, date, password, or email address

- **android:inputType** attribute sets input type, affects
  - What type of keyboard pops up for user
  - Behaviors such as is every word capitalized

EditText Widget in Android Studio Palette

- A whole section of the Android Studio palette dedicated to EditText widgets (or text fields)

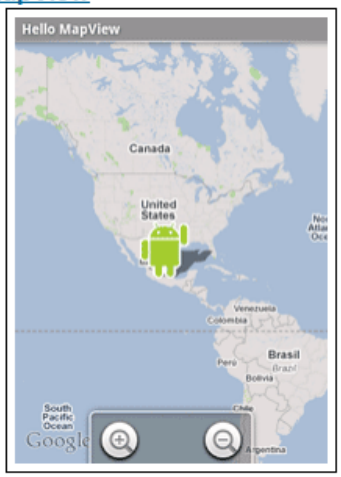


Text Fields
Section of Widget palette

EditText inputType menu

Other Available Widgets

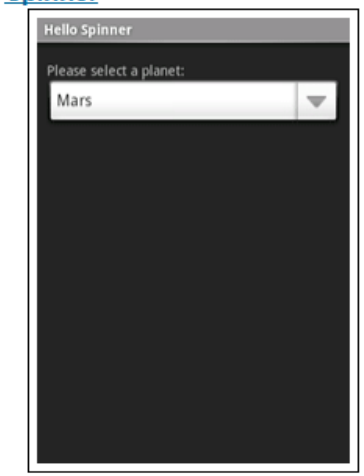
MapView



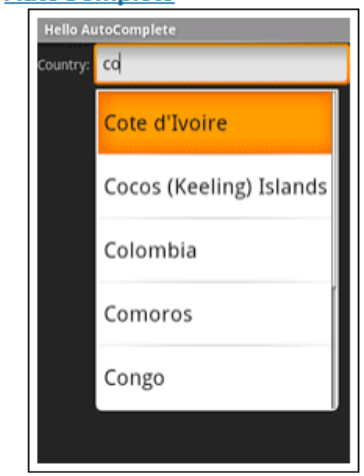
WebView

DatePicker

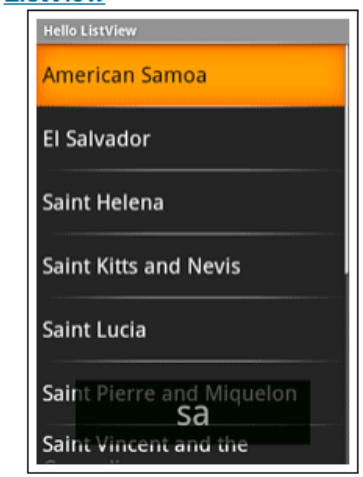
Spinner



AutoComplete



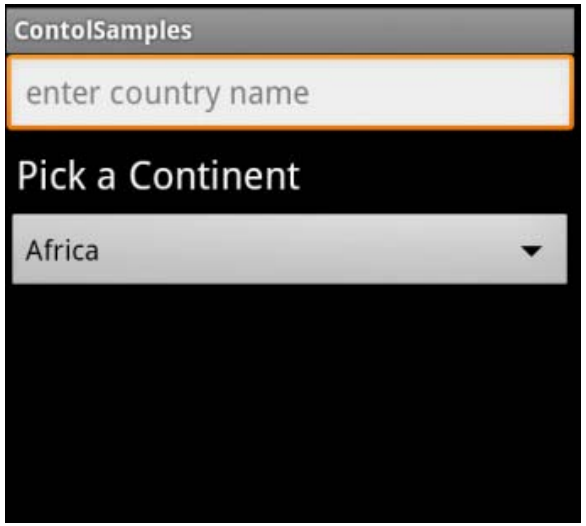
ListView

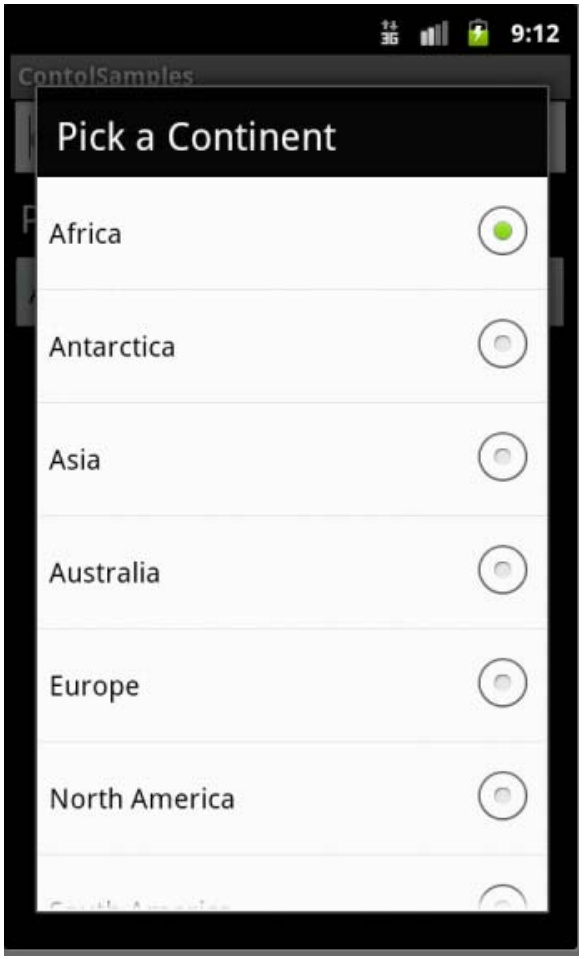


http://developer.android.com/resources/tutorials/views/index.html

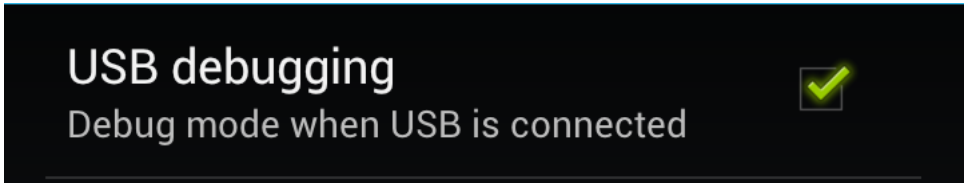
Spinner Controls

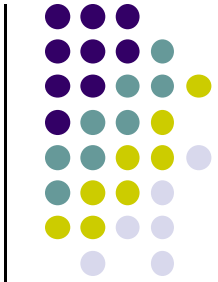
- Similar to auto complete, but user **must** select from a set of choices





Checkbox

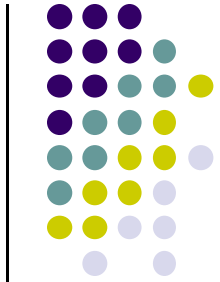




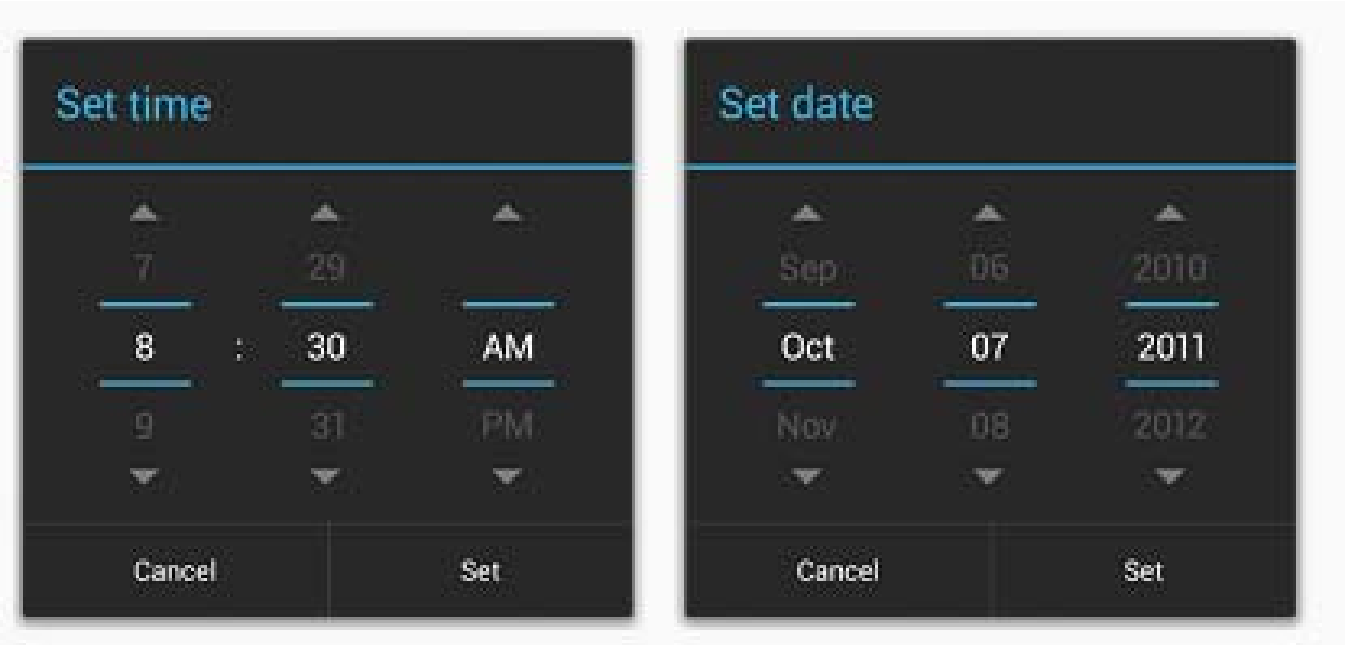
- Checkbox has 2 states: checked and unchecked
- Clicking on checkbox toggles between these 2 states
- Used to indicate a choice (e.g. Add rush delivery)
- Checkbox widget inherits from TextView, so its properties like android:textColor can be used to format checkbox
- XML code to create Checkbox

```
<?xml version="1.0" encoding="utf-8"?>
<CheckBox xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/check"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/unchecked"/>
```

Pickers

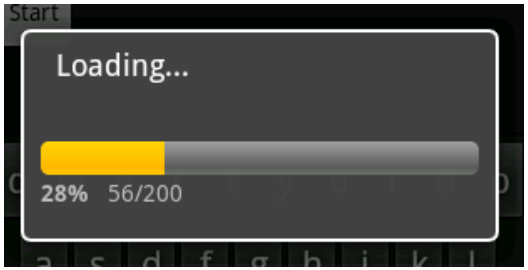


- TimePicker and DatePicker
- Typically displayed in a TimePickerDialog or DatePickerDialog
  - Dialogs are small pop-up windows that appear in front of the current activity

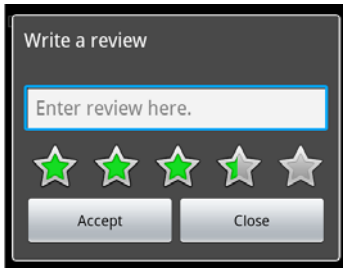


Indicators

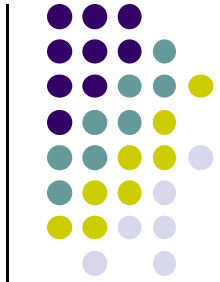
- Variety of built-in indicators in addition to TextView
- ProgressBar



- RatingBar

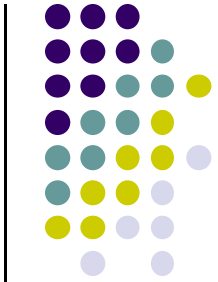


- Chronometer
- DigitalClock
- AnalogClock



Android UI Youtube Tutorials

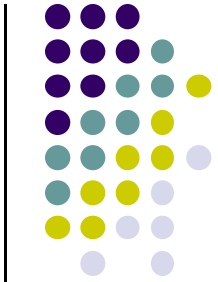
Tutorial 11: Designing the User Interface



- Tutorial 11: Designing the User Interface [6:19 mins]
  - https://www.youtube.com/watch?v=72mf0rmjNAA

- Main Topics
  - Designing the User interface
  - Manually adding activity
  - Dragging in widgets
  - Changing the text in widgets

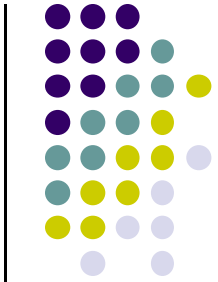
Tutorial 12: More on User Interface



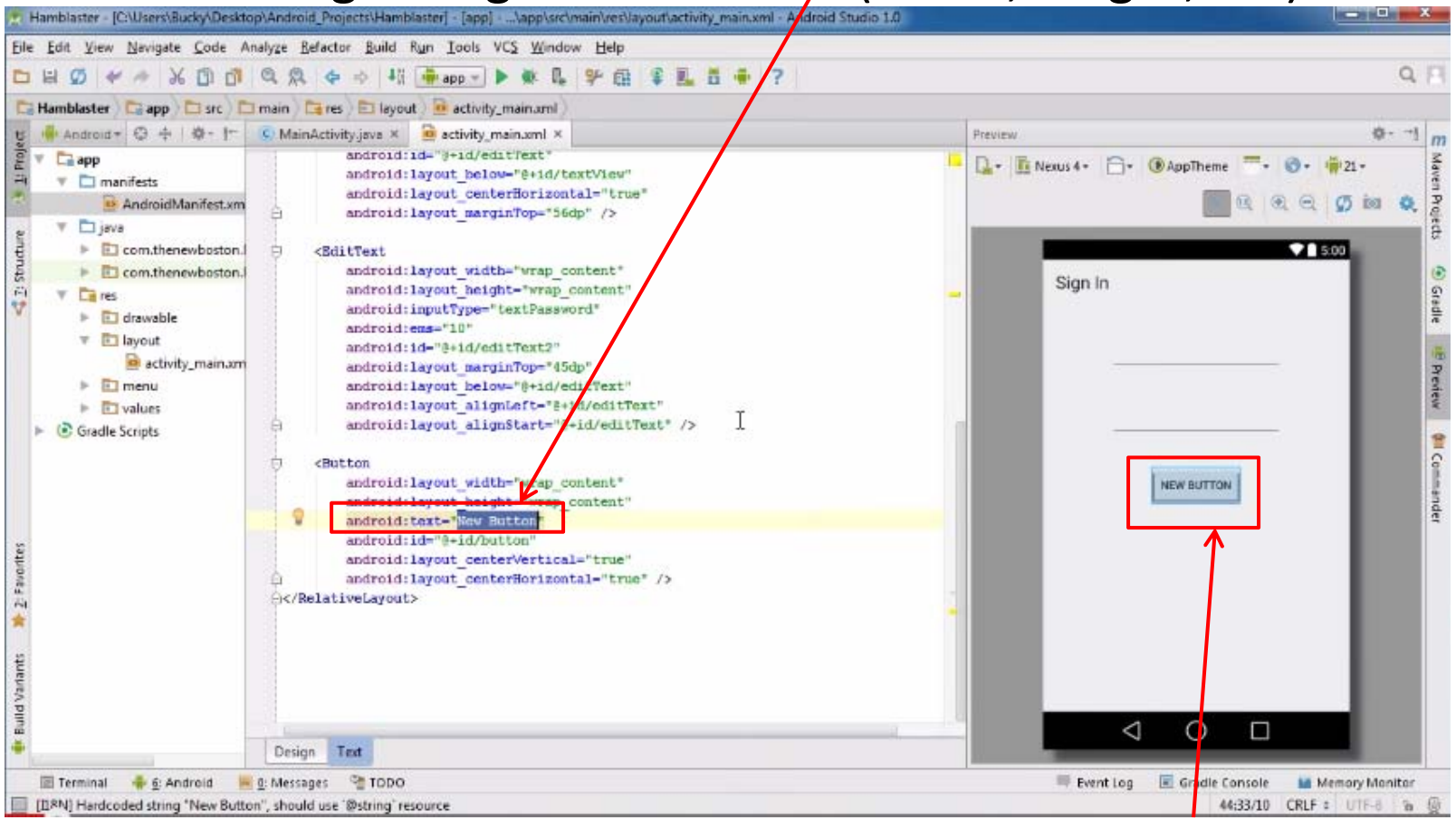
- Tutorial 12: More on User Interface [10:24 mins]
  - https://www.youtube.com/watch?v=72mf0rmjNAA

- Main Topics
  - Changing text in widgets
  - Changing strings from hardcoded to resources (variables)

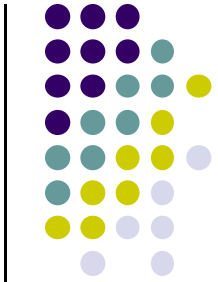
Changing Widget text in Text View



- E.g. Change text on New Button in activity_main.xml
- Can also change widget dimensions (width, height, etc)



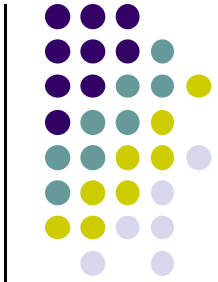
Tutorial 17: GridLayout



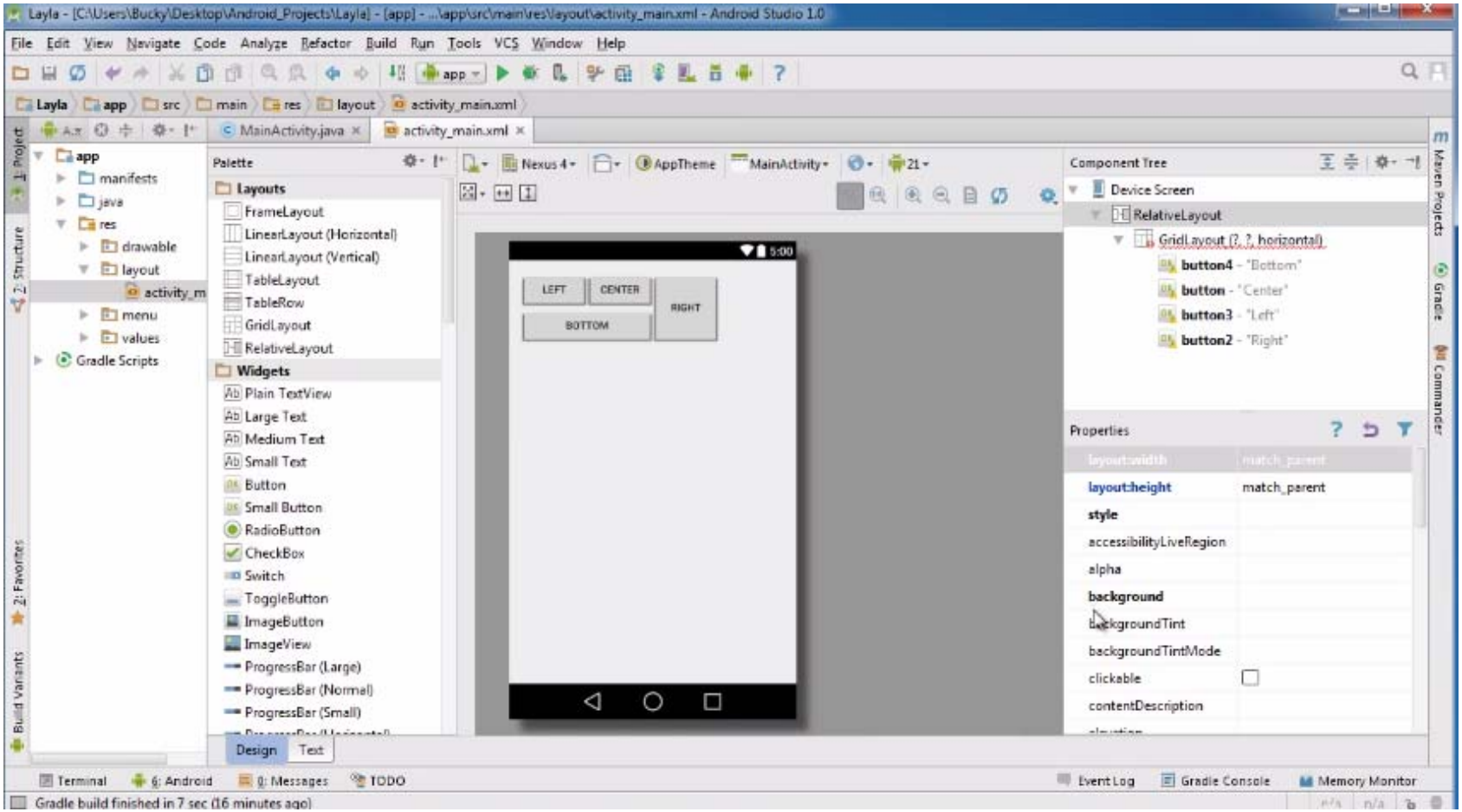
- Tutorial 17: GridLayout [9:40 mins]
  - https://www.youtube.com/watch?v=4bXOr5Rk1dk

- Main Topics
  - Creating GridLayout: Layout that places its children in a grid
  - Add widgets (buttons) to GridLayout
  - Format width, height, position of widgets

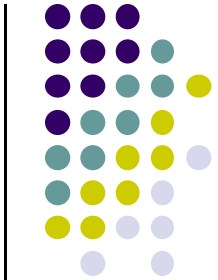
Create Grid Layout, Add & Format Widgets



- Add widgets (buttons) to GridLayout
- Format width, height, position of widgets



References



- Busy Coder's guide to Android version 4.4
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014
- Android App Development for Beginners videos by Bucky Roberts (thenewboston)