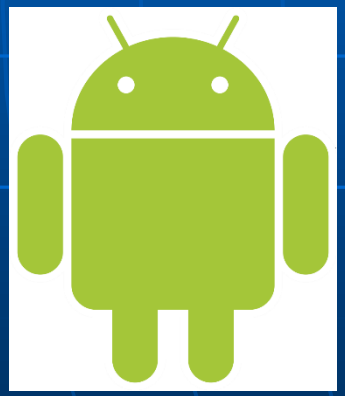
Mobile App Development

Android User Interfaces



Dr. Christelle Scharff
cscharff@pace.edu
Pace University, USA

Objectives

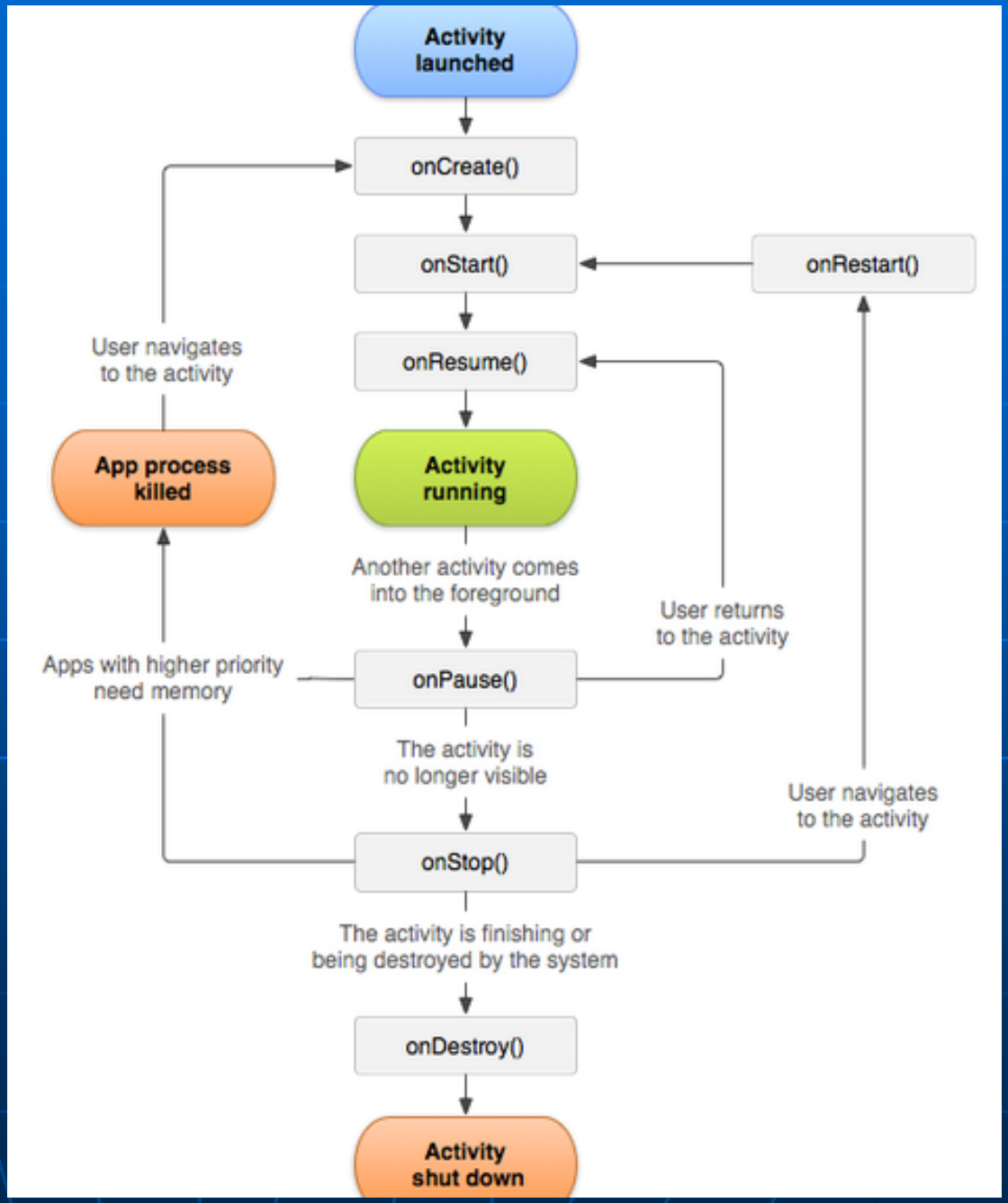
- Get familiar with Android layouts and building blocks
- Introduction to fragments and their life cycle
- Understand why XML is the standard way to design interfaces in Android
- Practice with and design simple layouts
- Get more comfortable with Android Studio

- Use layouts in Java
- Practice with simple building blocks (e.g., buttons)
- Understand and use basic UI controls (e.g. listeners)
- Introduction to Toast
- Introduction to Log

Activity

- An *Activity* creates a window in which UI components are placed
- Each screen is an *Activity*
- Activities are managed through the *ActivityManager*
- Check out the *Activity* and *ActivityManager* classes:
  - *http://developer.android.com/reference/android/app/Activity.html*
  - *http://developer.android.com/reference/android/app/ActivityManager.html*

Activity Life Cycle



Jetpack

Android Jetpack

- Jetpack is a set of components and tools to accelerate and architecture guidance the development of Android apps
- https://developer.android.com/jetpack
- Intro video: https://www.youtube.com/watch?v=LmkKFCfmnhQ&feature=youtu.be
- Backward compatibility with the support library
- Data persistence and changes with architecture components
- Guidance to onboard developers
- More maintainable apps
- 4 categories of components: Foundation, architecture, UI and behavior

Using Fragments

Warning

- The correct way to develop in Android is to use **Fragments**
- After API 11, Android 3.0
- To support more flexible and interactive UIs

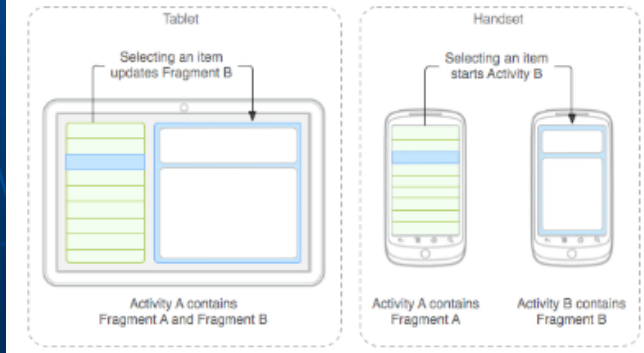


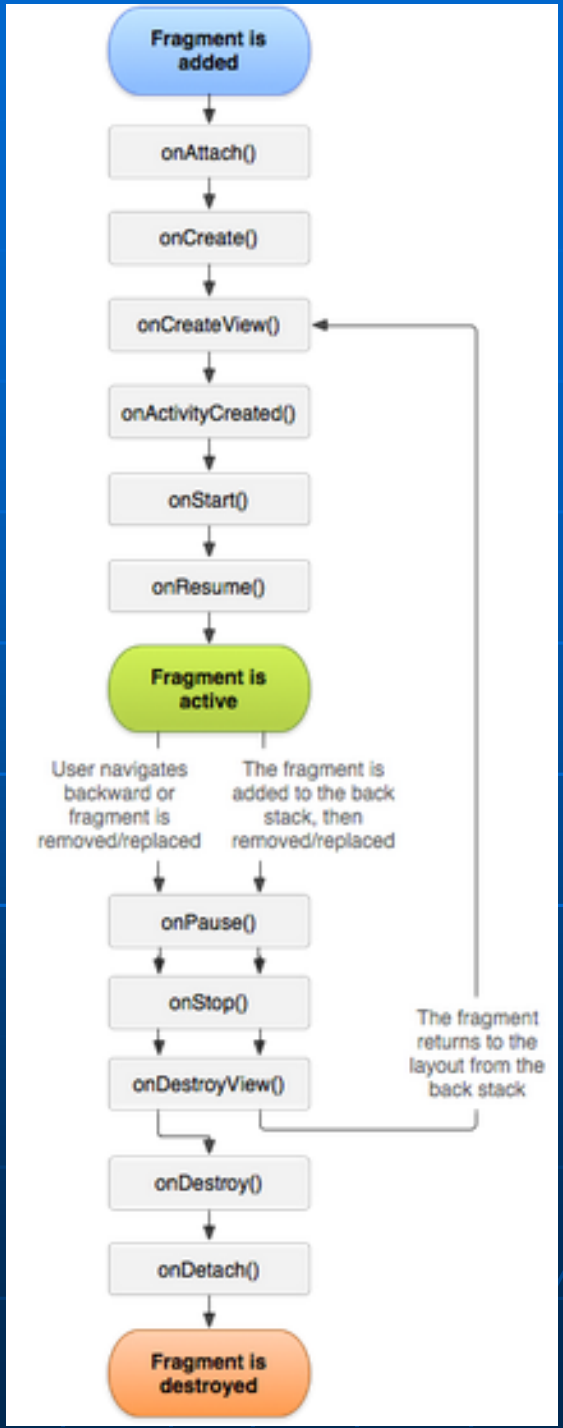
Figure 1. An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

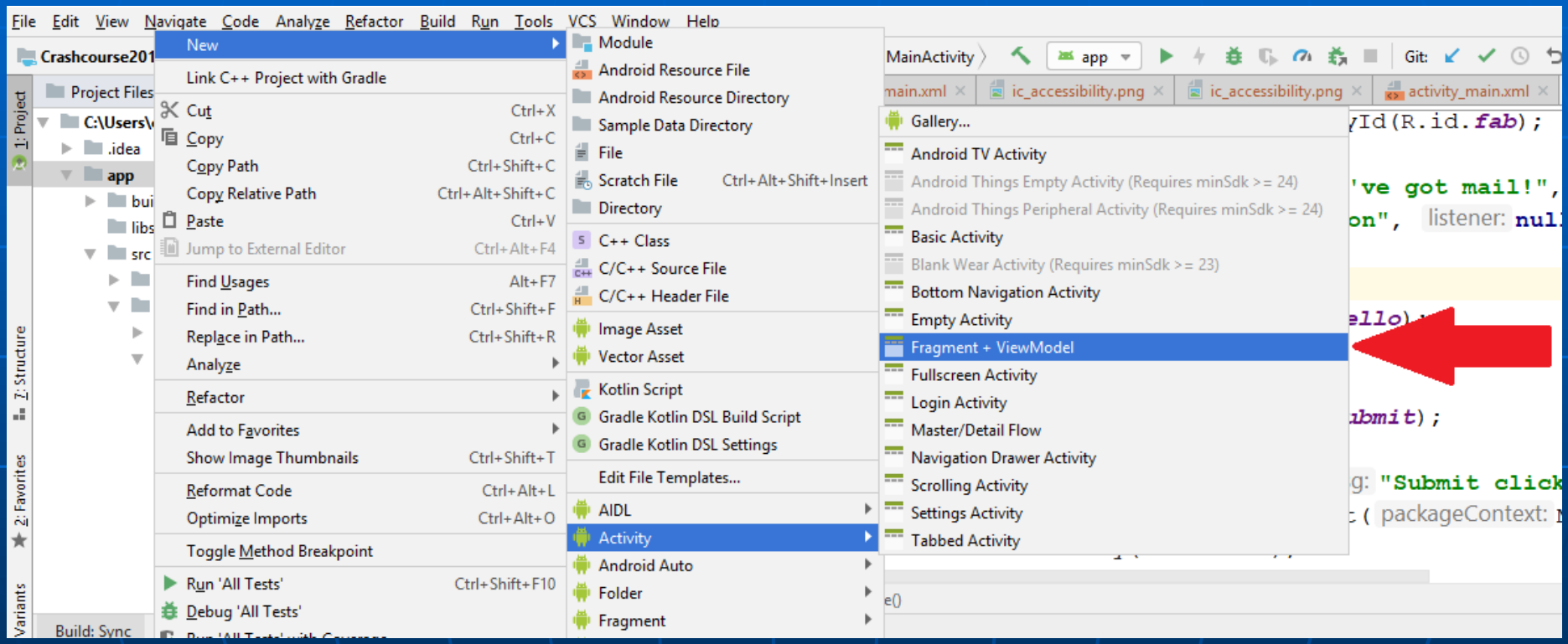
https://developer.android.com/guide/components/fragments#java

Fragments

- A *Fragment* is a piece of UI or behavior that can be placed in an activity or is independent
- Fragments are managed by FragmentManager
- Check out the *Fragment* and *FragmentManager* classes:
  - *http://developer.android.com/reference/android/app/Fragment.html*
  - *http://developer.android.com/reference/android/app/FragmentManager.html*
- A good video on fragments is available here:
  - https://www.youtube.com/watch?v=k3IT-IJ0J98

Fragment Life Cycle




File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Crashcourse201
Project Files
C:\Users\
.idea
app
src
Build: Sync
1: Project
7: Structure
2: Favorites
Variants
New
Link C++ Project with Gradle
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to External Editor Ctrl+Alt+F4
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails Ctrl+Shift+T
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Toggle Method Breakpoint
Run 'All Tests' Ctrl+Shift+F10
Debug 'All Tests'
Module
Android Resource File
Android Resource Directory
Sample Data Directory
File
Scratch File Ctrl+Alt+Shift+Insert
Directory
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Kotlin Script
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
Edit File Templates...
AIDL
Activity
Android Auto
Folder
Fragment
MainActivity
app
Git:
main.xml
ic_accessibility.png
ic_accessibility.png
activity_main.xml
Gallery...
Android TV Activity
Android Things Empty Activity (Requires minSdk >= 24)
Android Things Peripheral Activity (Requires minSdk >= 24)
Basic Activity
Blank Wear Activity (Requires minSdk >= 23)
Bottom Navigation Activity
Empty Activity
Fragment + ViewModel
Fullscreen Activity
Login Activity
Master/Detail Flow
Navigation Drawer Activity
Scrolling Activity
Settings Activity
Tabbed Activity
yId(R.id.fab);
've got mail!",
on", listener: null
ello);
ubmit);
g: "Submit click
(packageContext:
e()

Layouts

Layouts

- A layout is a specification of the interface, its building blocks, and their relationships to each other and their containers. It is written in XML
- Layouts are ViewGroups https://developer.android.com/reference/android/view/ViewGroup.html
- There are different types of layouts, e.g., ConstraintLayout, AbsoluteLayout, LinearLayout, RelativeLayout etc
  - *http://developer.android.com/guide/topics/ui/declaring-layout.html*
- Android provides a toolbox of standard building blocks to design UI interfaces, e.g., Button, CheckBox, RadioButton, Spinner, ProgressBar, Seekbar
  - *http://developer.android.com/reference/android/widget/package-summary.html*
- Widgets are Views https://developer.android.com/reference/android/view/View.html

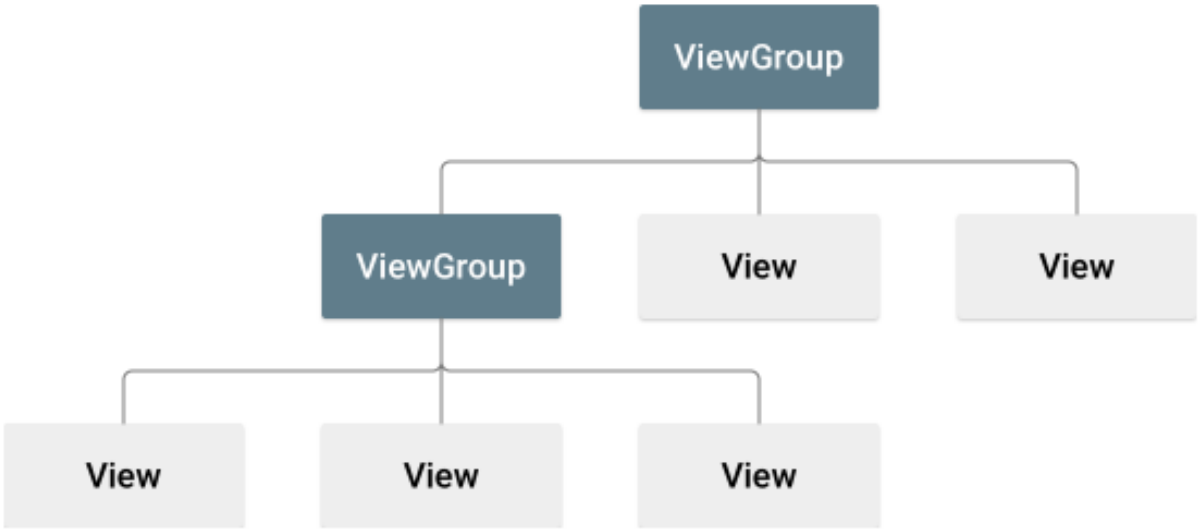


Figure 1. Illustration of a view hierarchy, which defines a UI layout

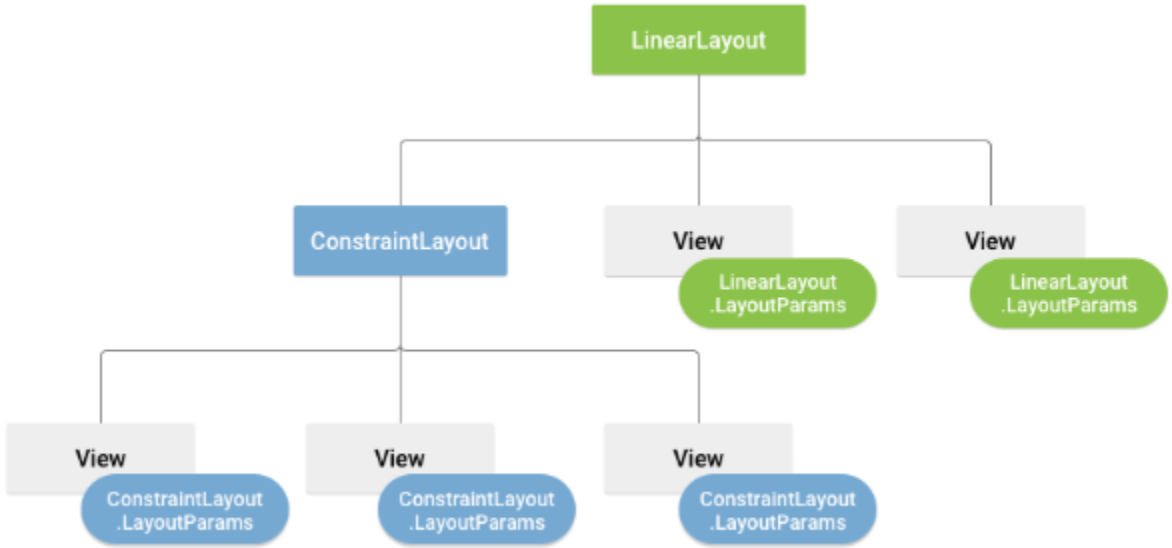
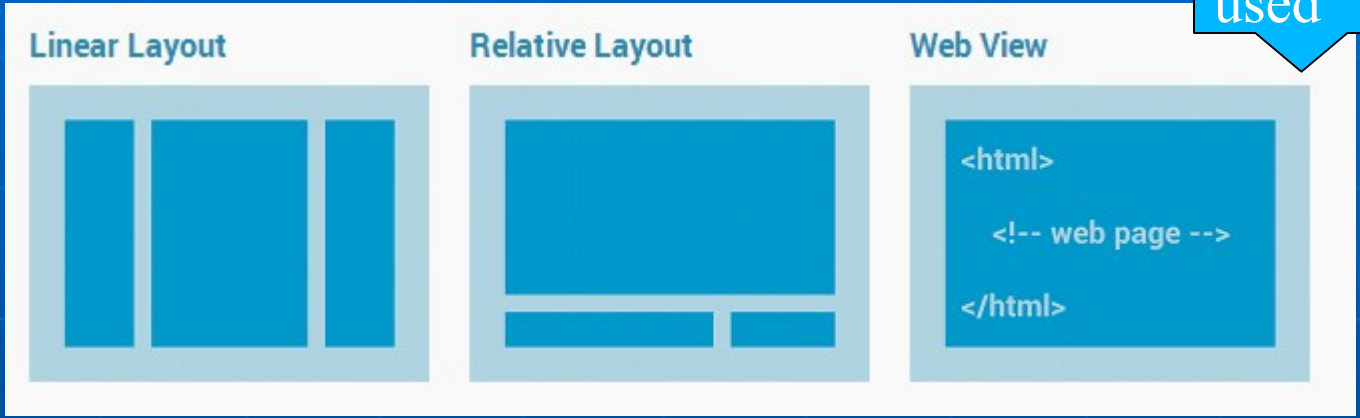
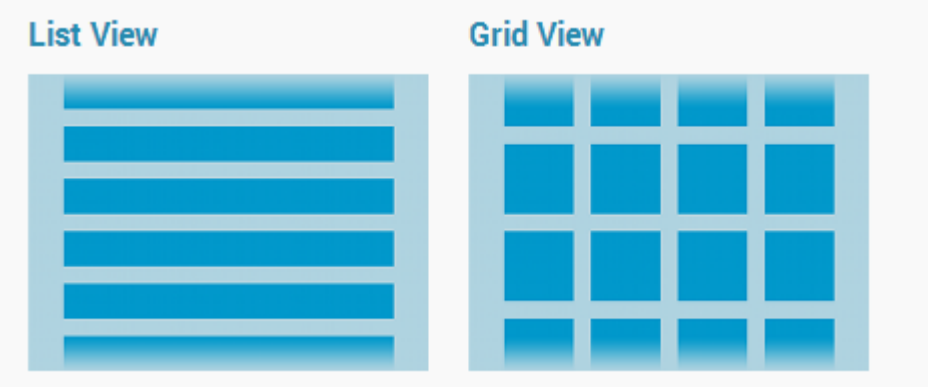


Figure 2. Visualization of a view hierarchy with layout parameters associated with each view

Common Layouts

When you do mobile web apps Web Views are used

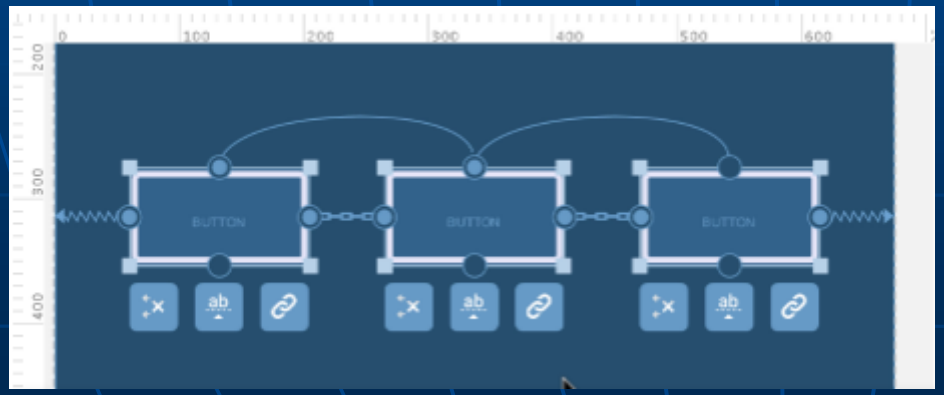




http://developer.android.com/guide/topics/ui/declaring-layout.html

New Layout: ConstraintLayout

- *ConstraintLayout* is similar to *RelativeLayout* but more flexible and easier to use with the Layout Editor. It does not require a nested view hierarchy
- Available after API 9 (Android 2.3)
- Constraints impact the positions
- https://developer.android.com/training/constraint-layout/index.html



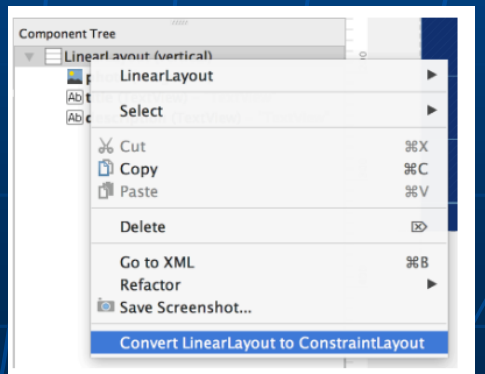
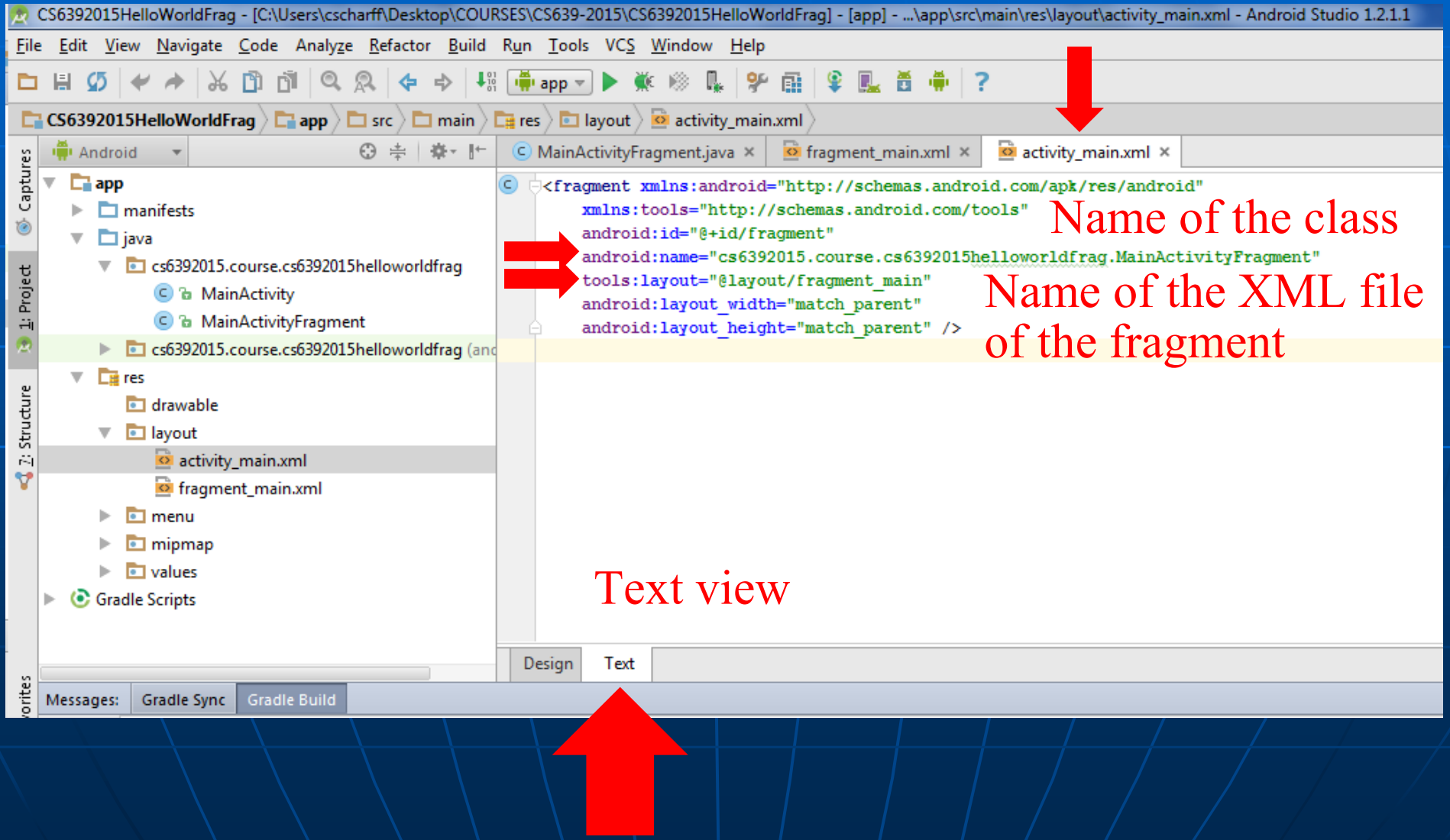


Figure 3. The menu to convert a layout to `ConstraintLayout`

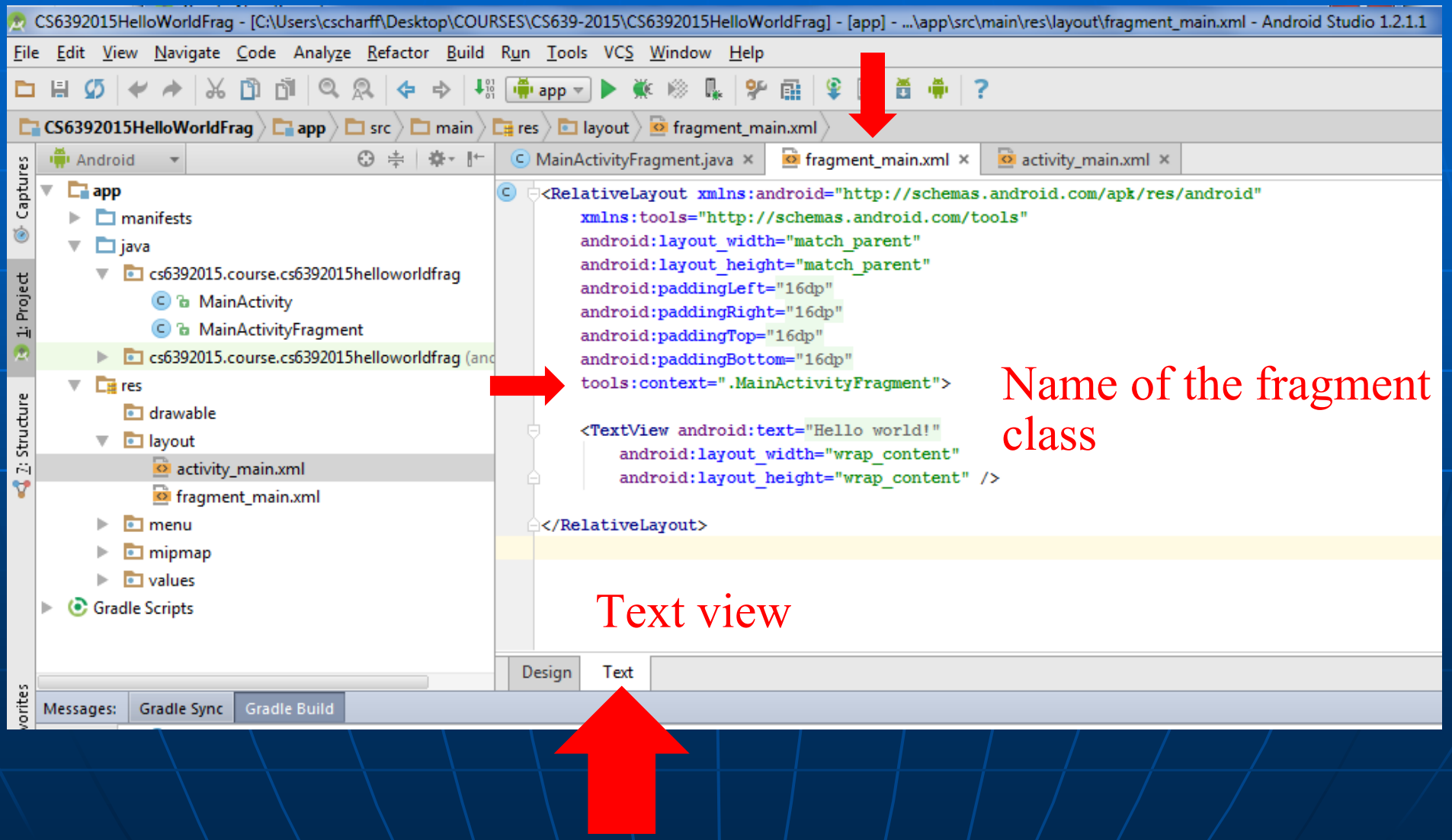
Question

- Think about use cases where you would use LinearLayout and GridView layouts

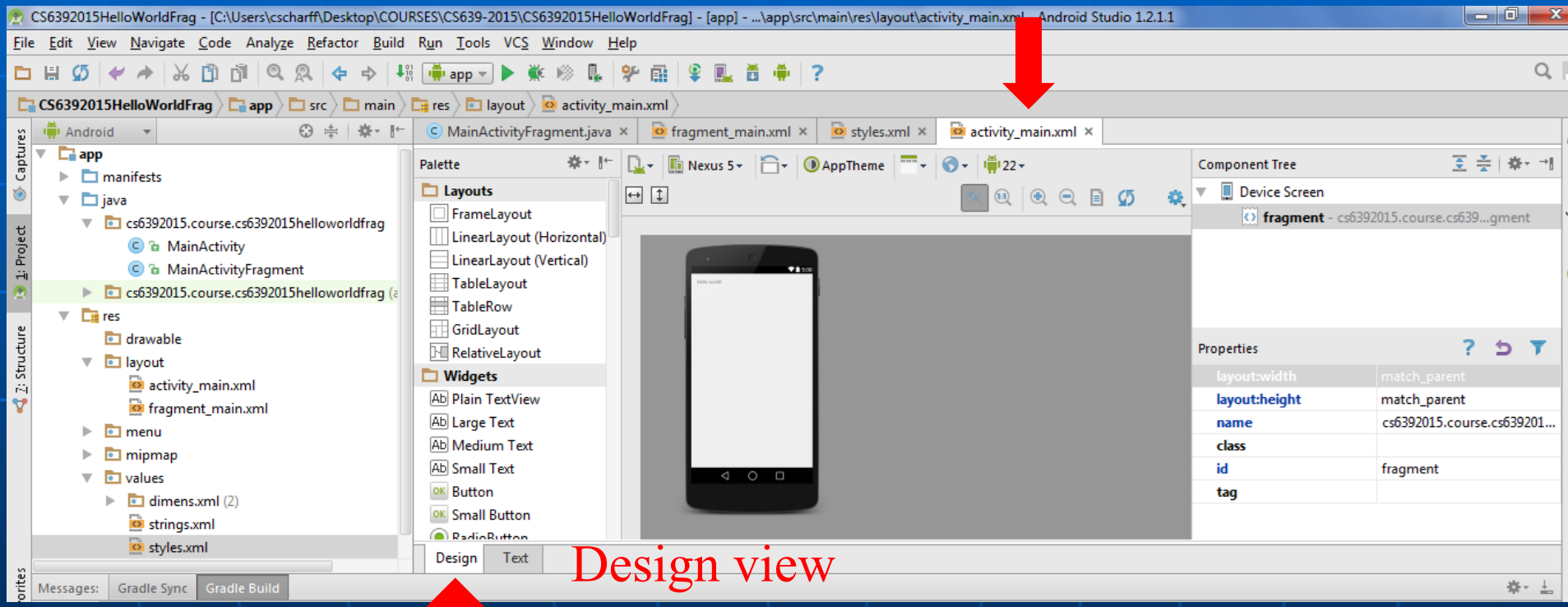
Example –activity_main.xml is a fragment



Example –fragment_main.xml contains the layout



Example – Text view of activity_main.xml



Design view

Lab

- Create a project with a fragment
- Locate the xml files
- In the Android Studio Designer locate the *Design* and the *Text* views of the layout
- Explore the main files and their code

Example – RelativeLayout with a TextView

```
<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"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivityFragment">

    <TextView android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

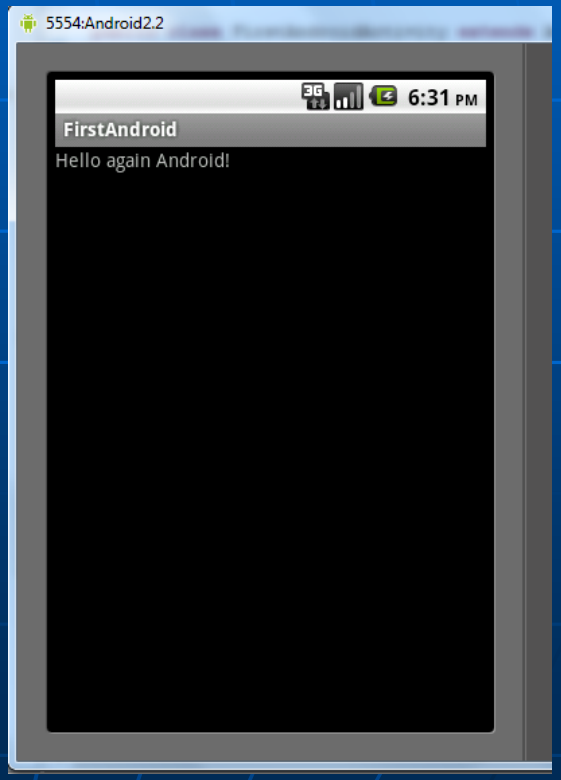
Why XML?

- XML-based layouts are resources saved in res/layout
- Why XML-based layouts?
  - To separate the user interface from the logic (**MVC** approach)
  - To have the **designers** do the layouts!
  - To have a **powerful resource resolution system** in place (tablets, phones etc.)
  - Have XML as a **standard** for UI definition format

Layouts in Java

- It is possible to design layout programmatically (like it was the case of Java ME) **but** the standard is to use XML!

```
package chris.android;

import android.app.Activity;

public class FirstAndroidActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tvHello = new TextView(this);
        tvHello.setText("Hello again Android!");
        setContentView(tvHello);
    }
}
```



Nested Layouts

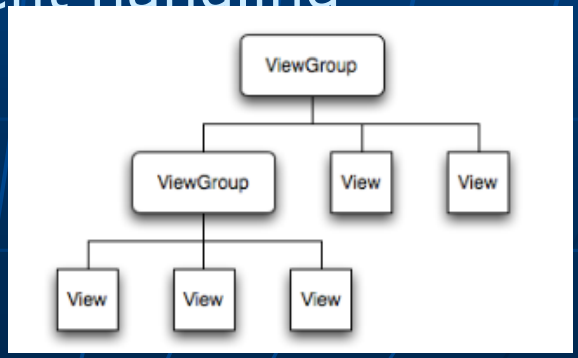
- A nested layout is a layout inside a layout
- Depth of the layouts has to be less than 8
- Do not use more than 10 nested layouts overall in an app
- Do not use more than 80 views overall in an app
- Bad design decisions challenge the rendering and responsiveness on phones (especially older and slower ones)

Views and their Properties

Views and ViewGroups

- Layouts are viewgroups! Widgets are views!
- ViewGroups are invisible views that contain other views (their children)
  - *http://developer.android.com/reference/android/view/ViewGroup.html*
- Views are the basic building blocks of Android user interface components
  - *http://developer.android.com/reference/android/view/View.html*
- A View occupies a rectangular area on the screen and is responsible for drawing and event handling

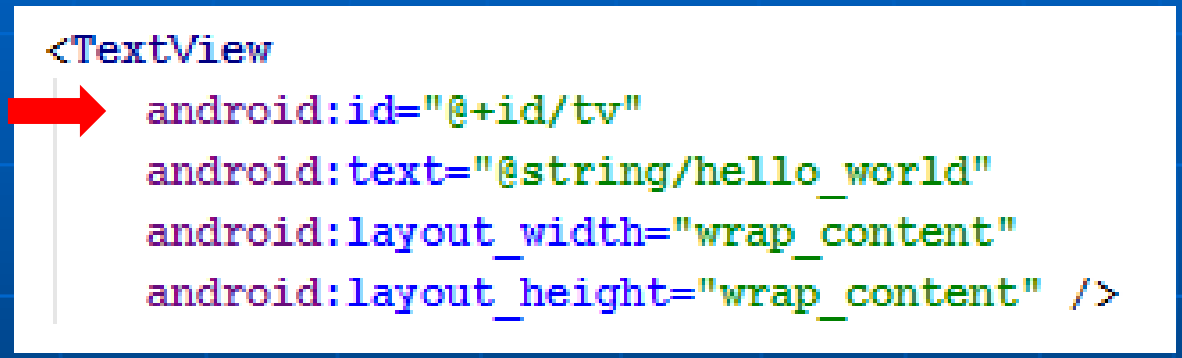
Nested viewgroups Parent / Child relationship



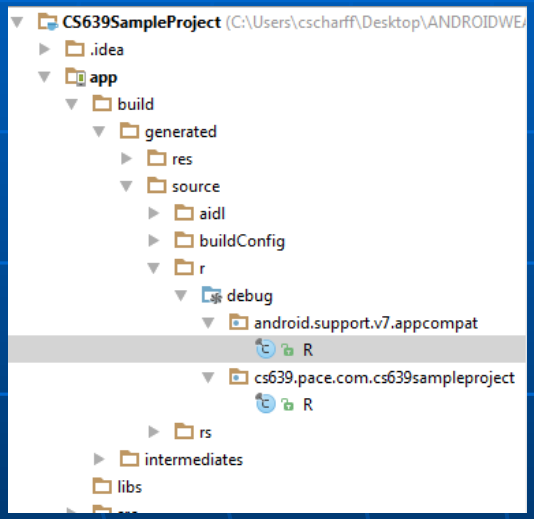
Views Definitions in XML

- Views have ids
  - `android:id` – Set the id of a view
  - Example: `android:id="@+id/title"`
  - Ids are maintained in `R.java (autogenerated file)`
- Views have properties
  - `android:text` – Set the text of the view
  - `android:visibility` – Set the visibility of the view (visible, invisible, gone)
  - `android:background` – Set the RGB color value of the view (e.g., #00FF00)
  - `android:layout_width` and `android:layout_height` – Set the size of the view

TextView: id and Properties



R.Java File



MyActivity.java × | R.java × | my.xml × | fragment_my.xml × | R.java × | app × | activity_my.xml ×

Files under the build folder are generated and should not be edited.

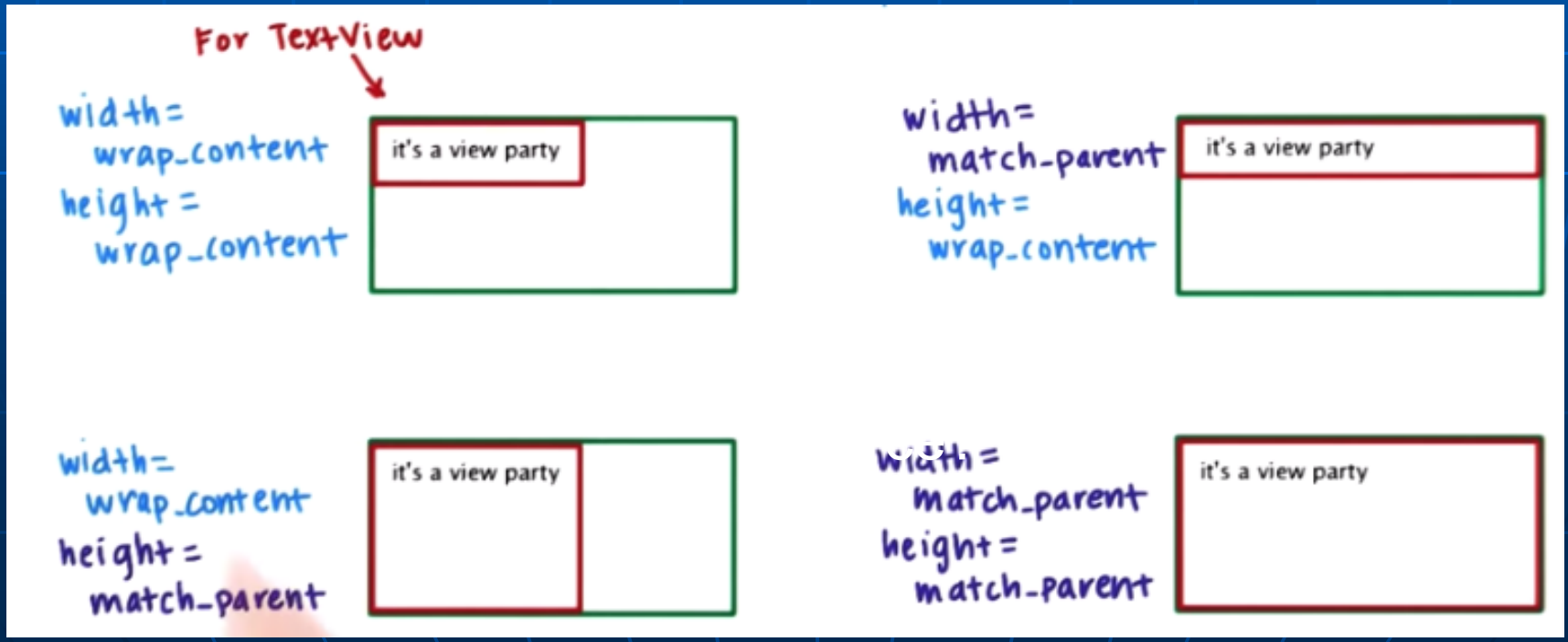
```
/.../
package android.support.v7.appcompat;

public final class R {
    public static final class anim {
        public static final int abc_fade_in = 0x7f040000;
        public static final int abc_fade_out = 0x7f040001;
        public static final int abc_slide_in_bottom = 0x7f040002;
        public static final int abc_slide_in_top = 0x7f040003;
        public static final int abc_slide_out_bottom = 0x7f040004;
        public static final int abc_slide_out_top = 0x7f040005;
    }
    public static final class attr {
        public static final int actionBarDivider = 0x7f010000;
        public static final int actionBarItemBackground = 0x7f010001;
        public static final int actionBarSize = 0x7f010002;
        public static final int actionBarSplitStyle = 0x7f010003;
        public static final int actionBarStyle = 0x7f010004;
        public static final int actionBarTabBarStyle = 0x7f010005;
        public static final int actionBarTabStyle = 0x7f010006;
        public static final int actionBarTabTextStyle = 0x7f010007;
        public static final int actionBarWidgetTheme = 0x7f010008;
        public static final int actionButtonStyle = 0x7f010009;
        public static final int actionDropDownStyle = 0x7f010066;
```

If build does not appear, locate the file directly in the file system or change the view of the project

Height and Width: match_parent / wrap_content

- Views always have a width and a height
  - dp (pixel density)
  - px (pixel) – Do not use pixels!
  - match_parent
  - wrap_content

Source?



gravity, layout_gravity and layout_weight properties

- `android:gravity` – To place content in a view
  - Example: center, center_vertical, center_horizontal
  - *http://developer.android.com/reference/android/view/Gravity.html*
- `android:layout_gravity` – To place a child with respect to its parents
  - *http://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html#attr_android:layout_gravity*
- `android:layout_weight` – To set the space a view will take on the screen (in the parent)
  - Example: `android:layout_weight="1“`
  - *http://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html*

layout_weight example

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

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Prev"
            android:layout_weight="1"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Welcome!"
            android:layout_weight="1"
            android:gravity="center_horizontal"/>

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Next"
            android:layout_weight="1"/>
    </LinearLayout>
</LinearLayout>
```

prevnext.xml

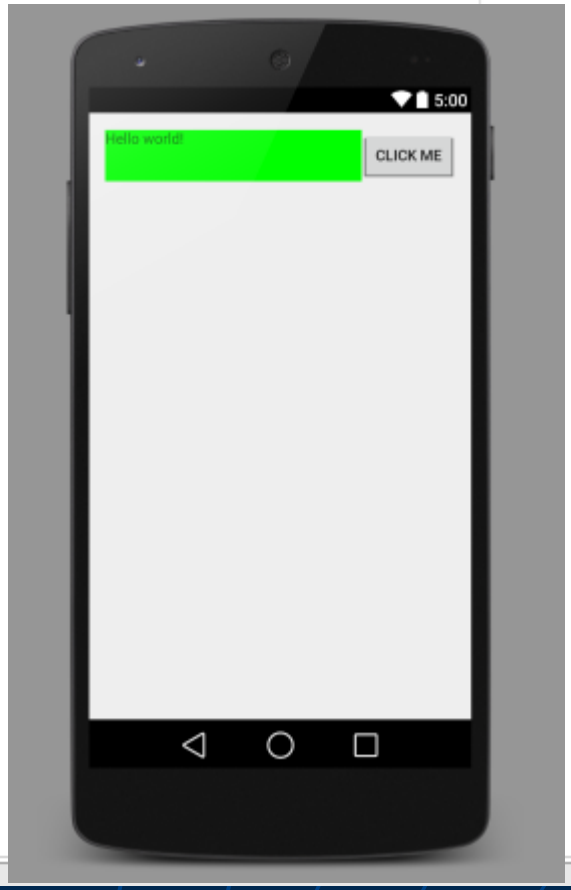
Nested linear layouts – horizontal and vertical

Code in GitHub

Example of Layout

MainActivityFragment.java | fragment_main.xml | prevnext.xml | app | styles.xml | activity_main.xml

```
<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"
    android:paddingLeft="@dimen/activity_horizontal_margin" android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin" android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivityFragment">
    <View
        android:id="@+id/myview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#00FF00"
        android:layout_alignParentTop="true"
        android:layout_alignBottom="@+id/btn"
        android:layout_toLeftOf="@id/btn"
        />
    <TextView
        android:id="@+id/title"
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        />
    <Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:text="Click me"/>
</RelativeLayout>
```

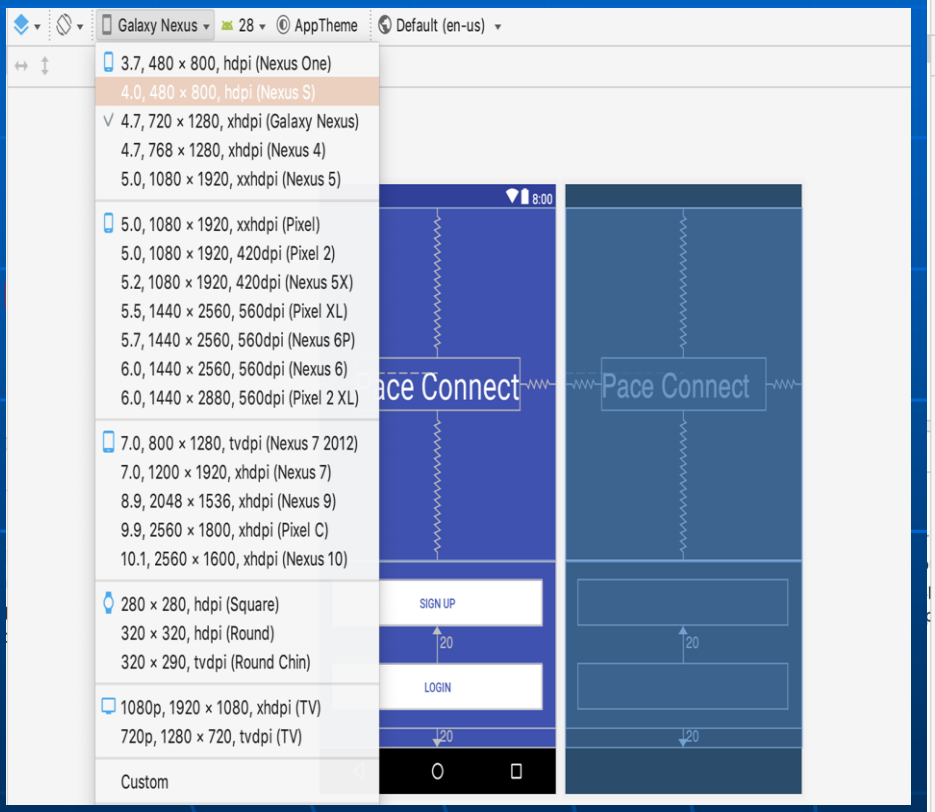


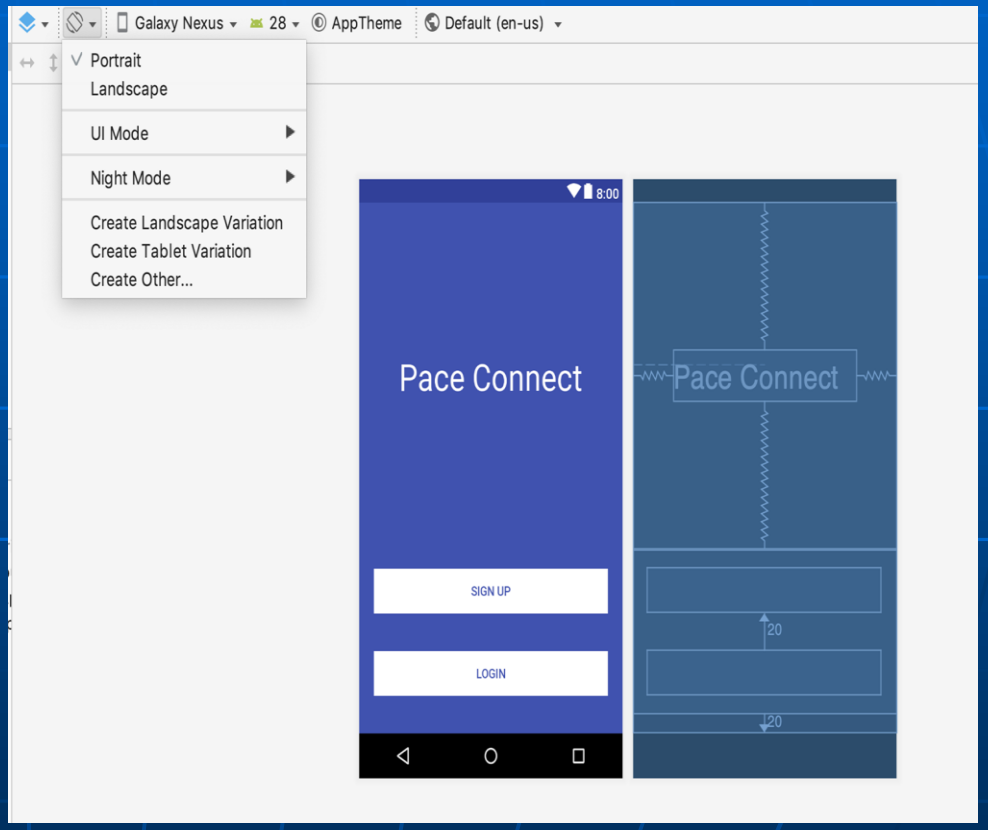
Questions

- Cite 5 properties of Views
  - *http://developer.android.com/reference/android/view/View.htm*

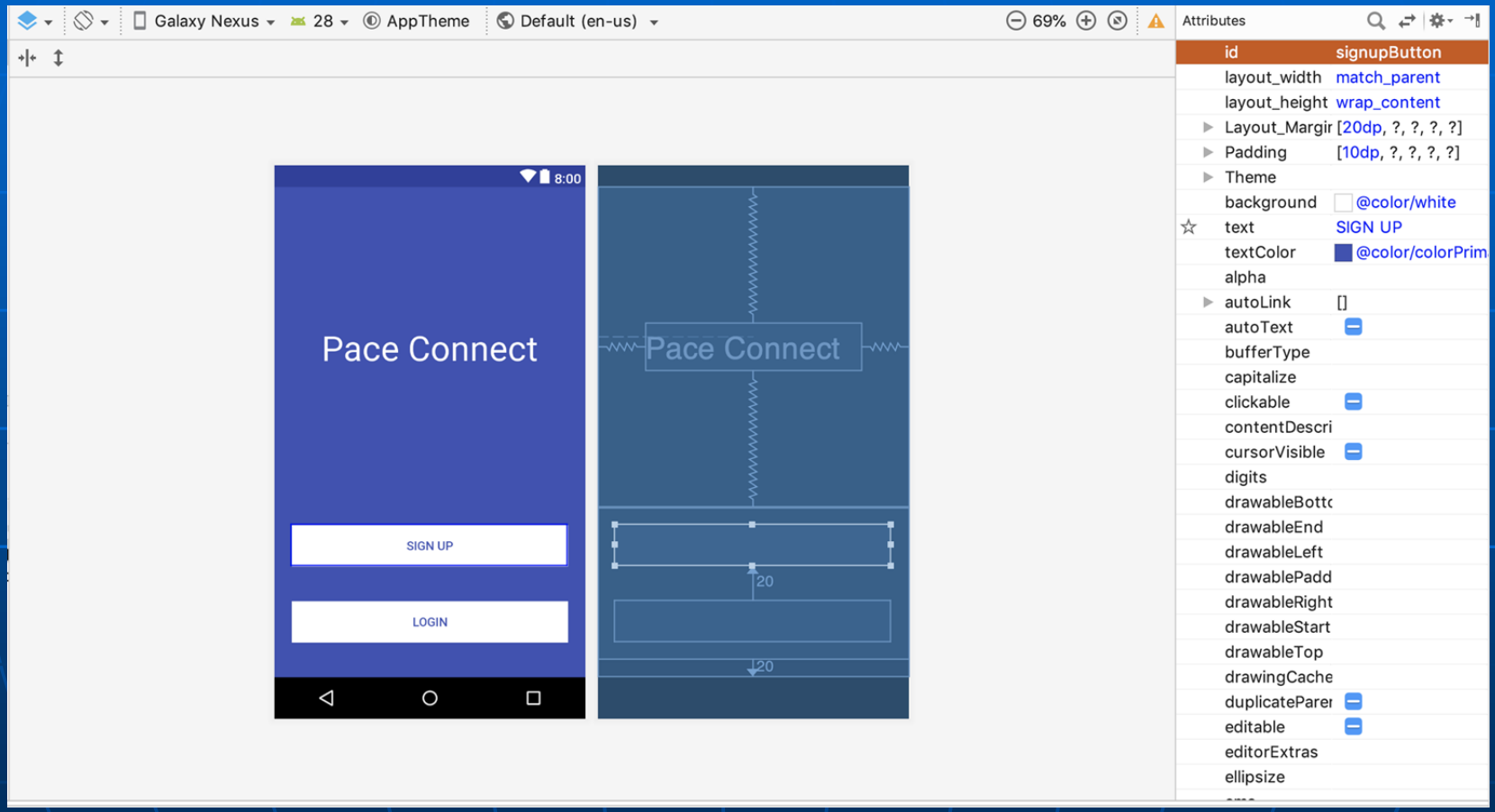
UI Designer in Android Studio

Using the Design View: Different Devices and Orientations





Building a Layout using the Design View



Drag & Drop

Properties of the selected component, here RelativeLayout

Lab

- Practice with the Android Studio UI designer interface, drag and drop some components, and change the properties of these components

LinearLayout

Linear Layout

- Each view is sized and placed dynamically on the screen in the order they are specified
- Vertically or horizontally
  - Vertically by default
  - `android:orientation="horizontal"`
  - `android:orientation="vertical"`
- Not appropriate for putting views exactly where you want them to be!

Linear Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

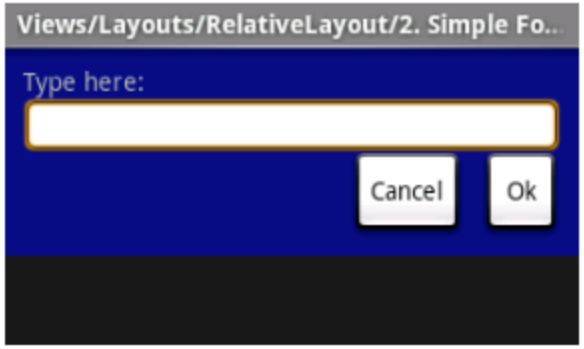
http://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts

RelativeLayout

Relative Layout

Queen of the screen layout system

```
<?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:background="@drawable/blue"
                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_below="@id/entry"
            android:layout_alignParentRight="true"
            android:layout_marginLeft="10px"
            android:text="OK" />

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



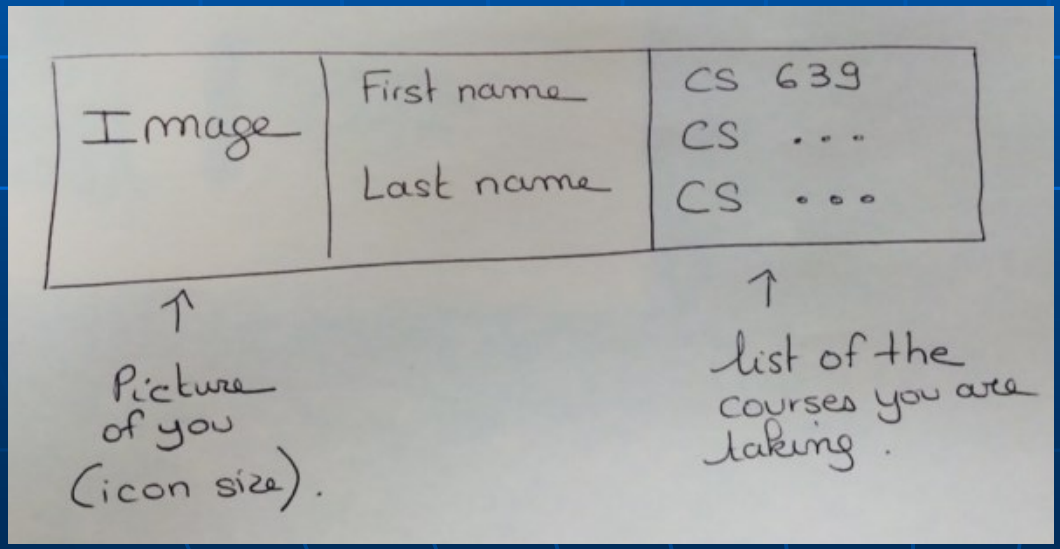
layout_below
layout_alignParentRight
layout_alignTop
layout_toLeftOf

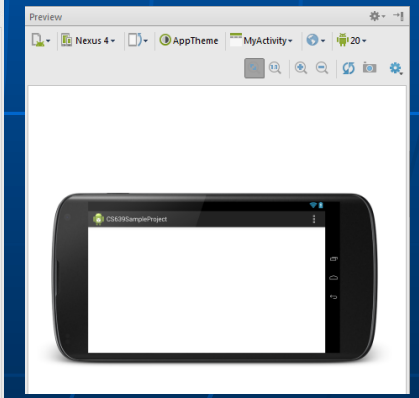
Elements are placed w.r.t each others

http://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts

Lab

- Create the layout depicted on the picture below
- Visualize the XML code in the UI designer in landscape and portrait
- What do you notice? Why?

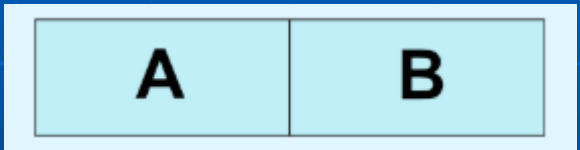




ConstraintLayout

ConstraintLayout

- Each view is sized and placed on the screen based on the elicitated constraints



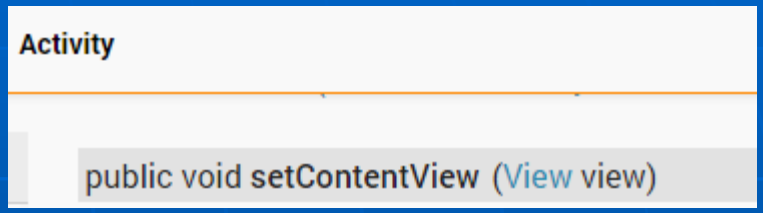

```
<Button android:id="@+id/buttonA" ... />
        <Button android:id="@+id/buttonB" ...
                app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```

- layout_constraintLeft_toLeftOf
- layout_constraintLeft_toRightOf
- layout_constraintRight_toLeftOf
- layout_constraintRight_toRightOf
- layout_constraintTop_toTopOf
- layout_constraintTop_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintBottom_toBottomOf
- layout_constraintBaseline_toBaselineOf
- layout_constraintStart_toEndOf
- layout_constraintStart_toStartOf
- layout_constraintEnd_toStartOf
- layout_constraintEnd_toEndOf

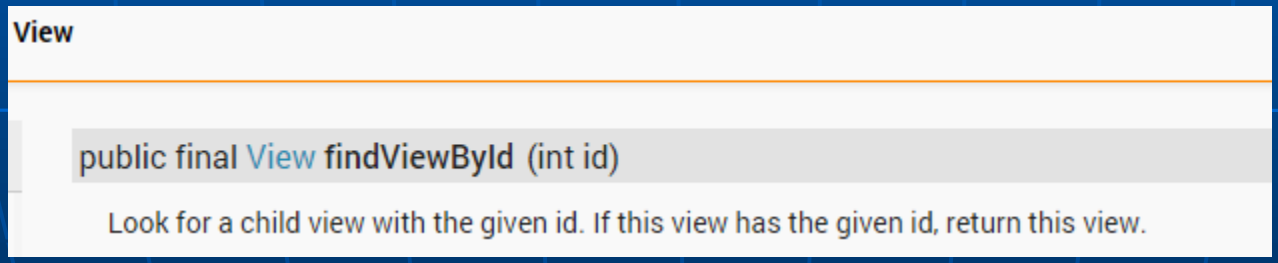
Access of XML from Java

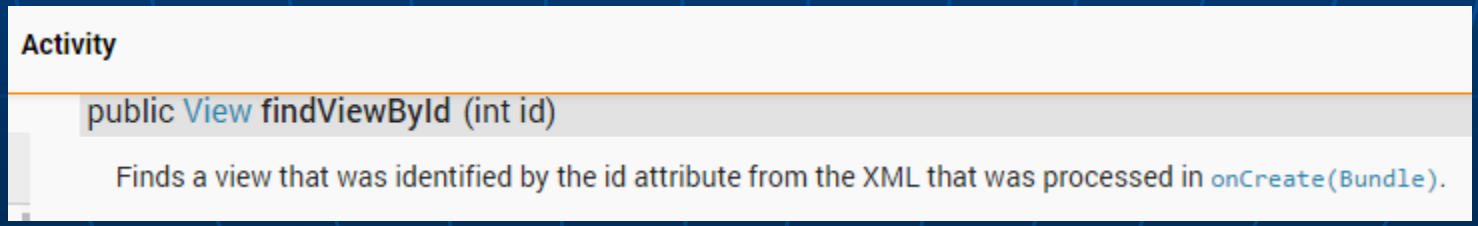
Accessing a View from the Java Code

- setContentView
  - Set an activity content to a view



- findViewById
  - Use the view identifier (id) to retrieve the view uniquely
  - View.findViewById() or Activity.findViewById()





http://developer.android.com/reference/android/app/Activity.html

```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

       FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener((view) -> {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
```

This is the floating button

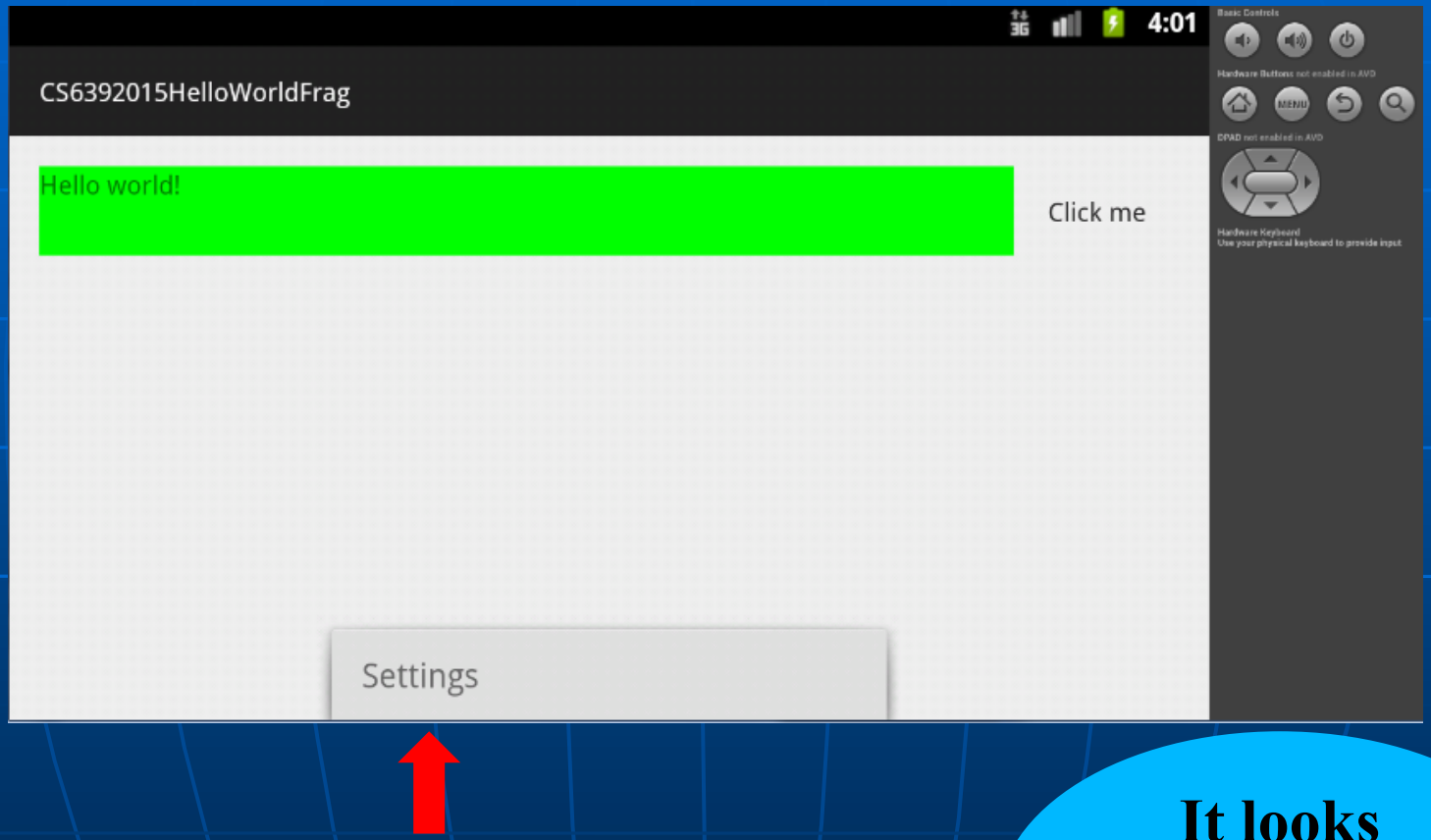
Explanation of the Code

- Java XML
- *MainActivity.java* *activity_main.xml*

- Methods:
  - onCreate
    - Launch the activity
    - R.layout.activity_main is the layout of the main activity that will be displayed
  - onCreateOptionsMenu
    - Inflates the *R.menu.menu_main* file
    - Inflates means that the XML file is transformed to be accessible from Java
  - onCreateOptionsMenuSelected
    - Is executed for each option (menu item) of the menu
    - Settings is an option by default

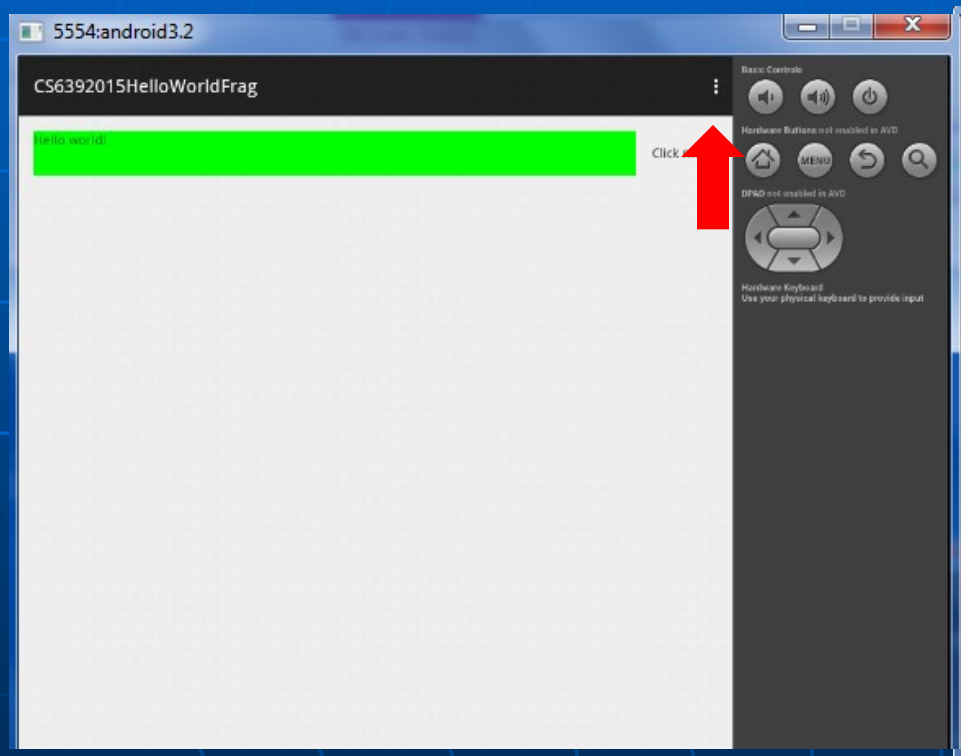
```java
/**
 * A placeholder fragment containing a simple view.
 */
public class MainActivityFragment extends Fragment {

    public MainActivityFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_main, container, false);
    }
}
```

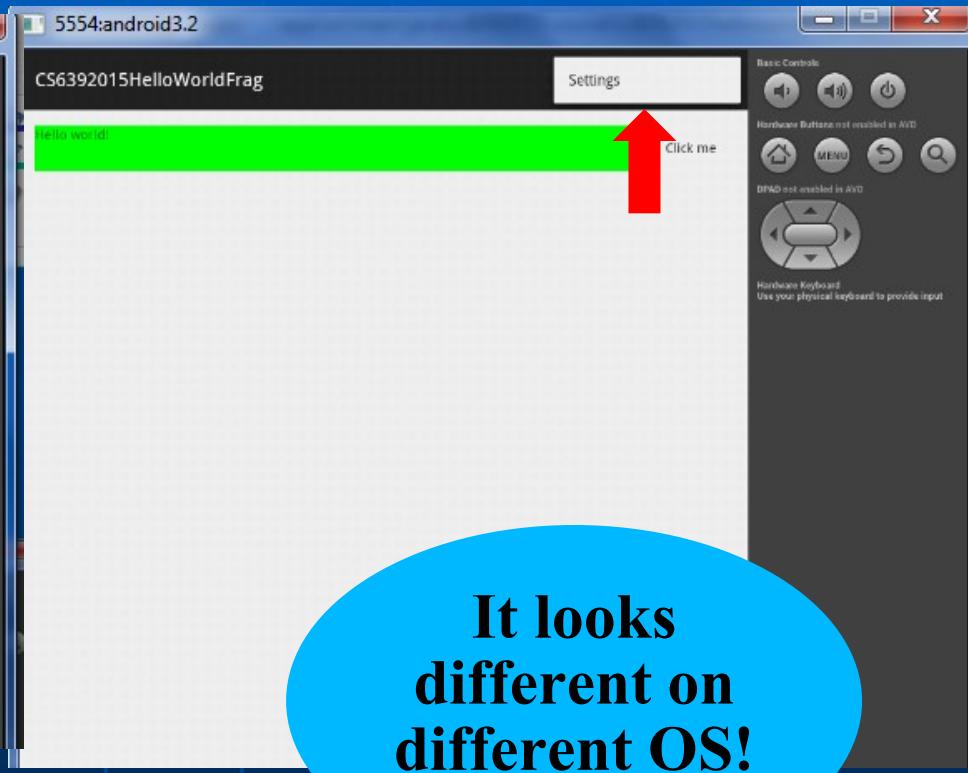
Execution of the Code on Android OS 2.3.3



It looks different on different OS!

Execution of the Code on Android OS 3.2





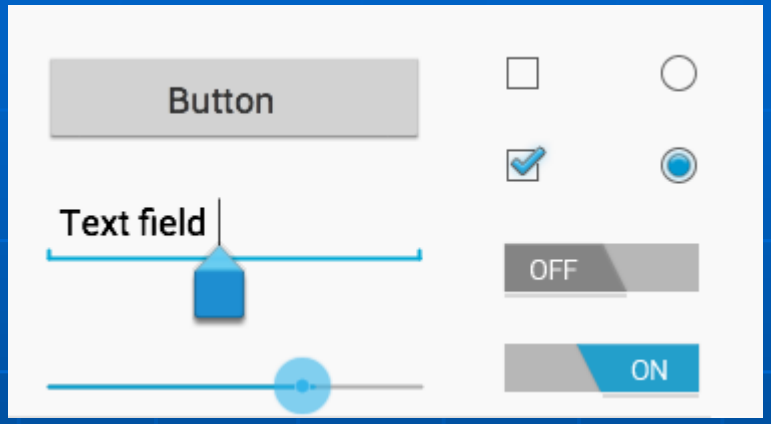
It looks different on different OS!

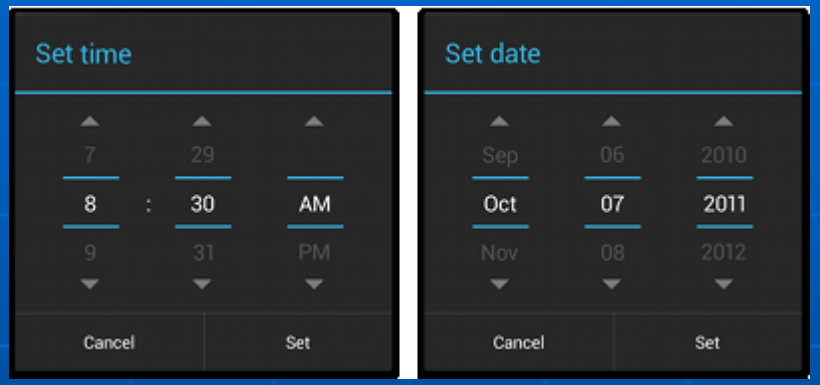
Questions

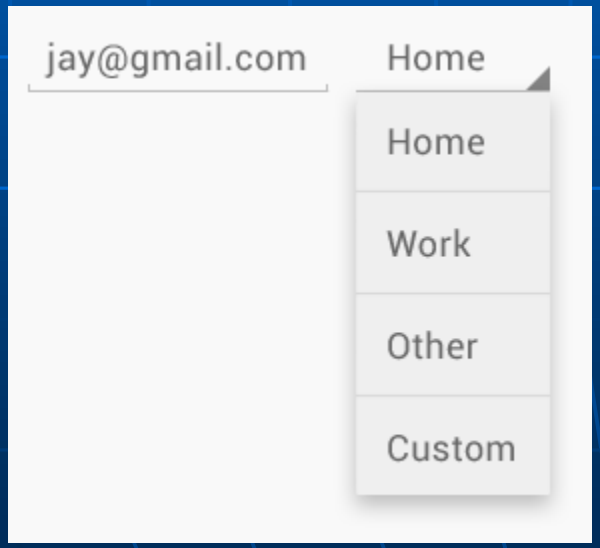
- Explore the *Activity* class
- What is *setContentView*?
- What is *findViewById*?
- *http://developer.android.com/reference/android/app/Activity.html*

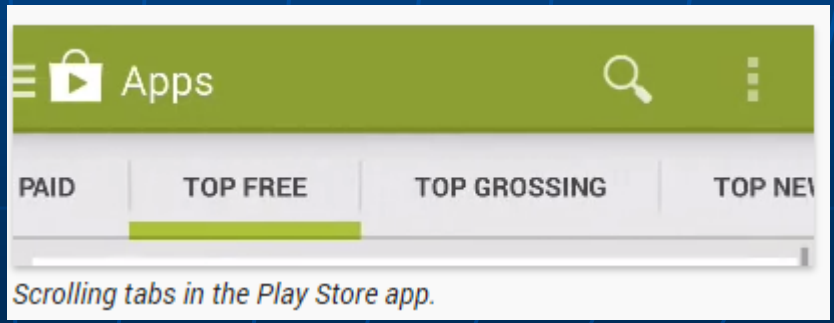
UI Components

Building Blocks









Scrolling tabs in the Play Store app.

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

TextView

TextView

- Labels (non editable texts) are referred to as *TextViews*
- Some XML properties of TextViews
  - Android:id - Identifier name for this view to be able to retrieve it with View.findViewById() or Activity.findViewById() (unique)
  - Android:layout_width – To set the width in pixels
  - Android:layout_height – To set the height in pixels
  - android:text – To set the text
  - android:typeface – To set the typeface to use for the label (e.g., normal, sans, serif, monospace)
  - android.:textStyle – To indicate if the typeface should be bold, italic or bold and italic (bold_italic)
  - android:textColor – Color of the label's text in RGB format (e.g., #FF0000 for red)
- More XML properties and info about the *TextView* class
  - *http://developer.android.com/reference/android/widget/TextView.html*

Example of TextView and Access from Java

- XML

```
<TextView android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="How are you?"
    android:id="@+id/TVquestion">
</TextView>
```

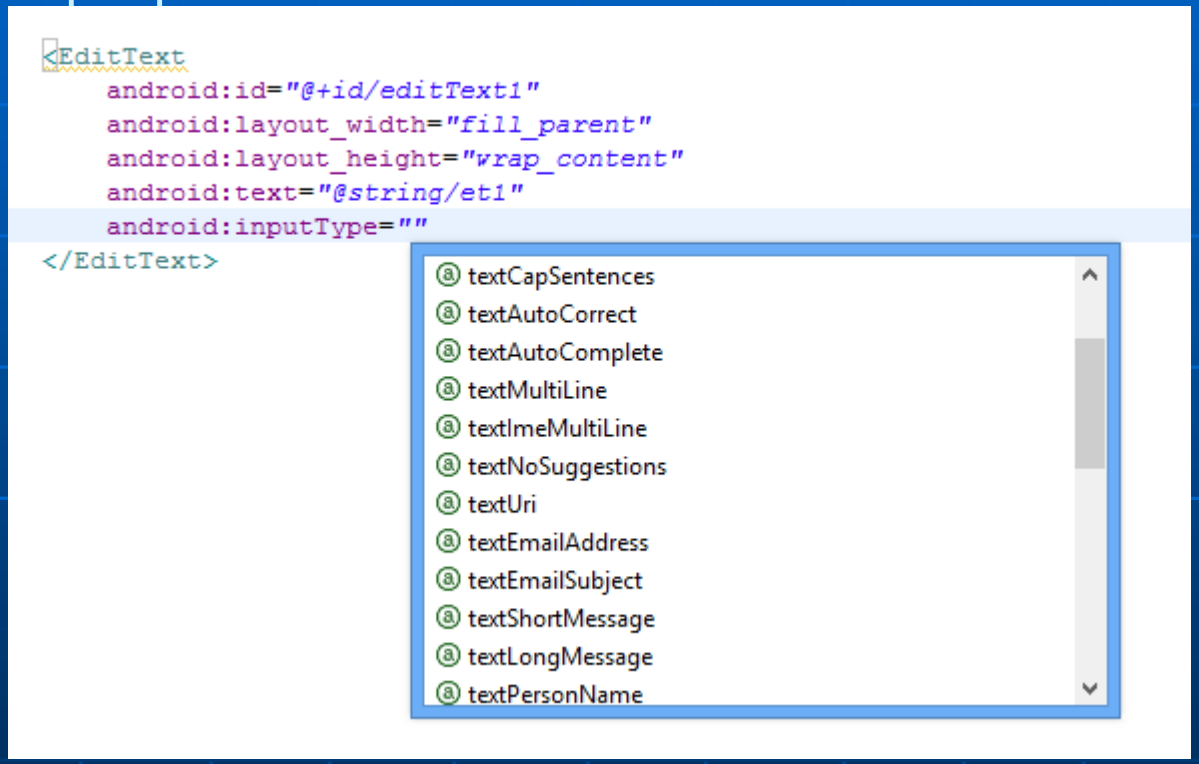
- Java

```
TextView label = (TextView)findViewById(R.id.TVquestion);
```

EditViews

EditText

- *EditText* is a subclass of *TextView* and is an editable label
- Some XML properties of *EditText*



- More XML properties and info about the Edit*View* class
  - *http://developer.android.com/reference/android/widget/EditText.html*

Button

Example of EditView and Access from Java

- XML

```
<EditText android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/ETName"
android:text="Dr. Scharff">
</EditText>
```

- Java

```
EditText txt = (EditText)findViewById(R.id.ETName);
```

```
txt.getText().toString()
```

For the content

Button

- *Button* is a subclass of *TextView.* A button can be pressed or pushed
- More XML properties and info about the *Button* class
  - *http://developer.android.com/reference/android/widget/Button.html*

XML

```
<Button android:layout_height="wrap_content"
android:layout_width="wrap_content" android:text="Click me!"
android:id="@+id/btnclickme"></Button>
</LinearLayout>
```

Java

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    btnClickMe = (Button) findViewById(R.id.btnclickme);
}
```

Listening to the Button

- Implementation with an anonymous inner class

```
package chris.android;

import android.app.Activity;

public class FirstAndroidActivity extends Activity {
    private Button btnClickMe;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btnClickMe = (Button) findViewById(R.id.btnclickme);
        btnClickMe.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Log.i("First Android Activity", "Click me clicked!!!");
                btnClickMe.setText("You clicked!");
            }
        });
    }
```

This code is without fragment. Never write this code by typing it but use CODE COMPLETION – CTRL + SPACE to generate the code

Listening to the Button

- Implementation without an anonymous inner class

```
package chris.android;

import android.app.Activity;

public class FirstAndroidActivity extends Activity implements OnClickListener {
    private Button btnClickMe;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btnClickMe = (Button) findViewById(R.id.btnclickme);
        btnClickMe.setOnClickListener(this);
    }

    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.btnclickme:
            Log.i("First Android Activity", "Click me clicked!!!");
            btnClickMe.setText("You clicked!");
        }

    }
}
```

Some Important Tools

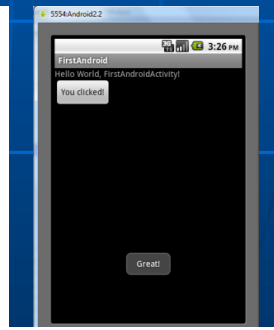
Toast

Toast

- A Toast is an unobtrusive floating view appearing on the top of an application
- It can be useful also for debugging purposes
- It is used to provide a message to the user and just uses the space required for the message

```
Context context = getApplicationContext();
CharSequence text = "Great!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Code in an activity




```
Toast t = Toast.makeText(getActivity().getApplicationContext(), "The button is clicked", Toast.LENGTH_L
t.show();
```

Code in an fragment

- *http://developer.android.com/reference/android/widget/Toast.html*

Do not forget show!

LogCat and Log

Logcat

- A mechanism for collecting, viewing and monitoring system **debug** output
- To filter the output, there are different options:
  - Verbose
  - Debug
  - Info
  - Warn
  - Error

http://developer.android.com/tools/help/logcat.html

Adding Logs in Java

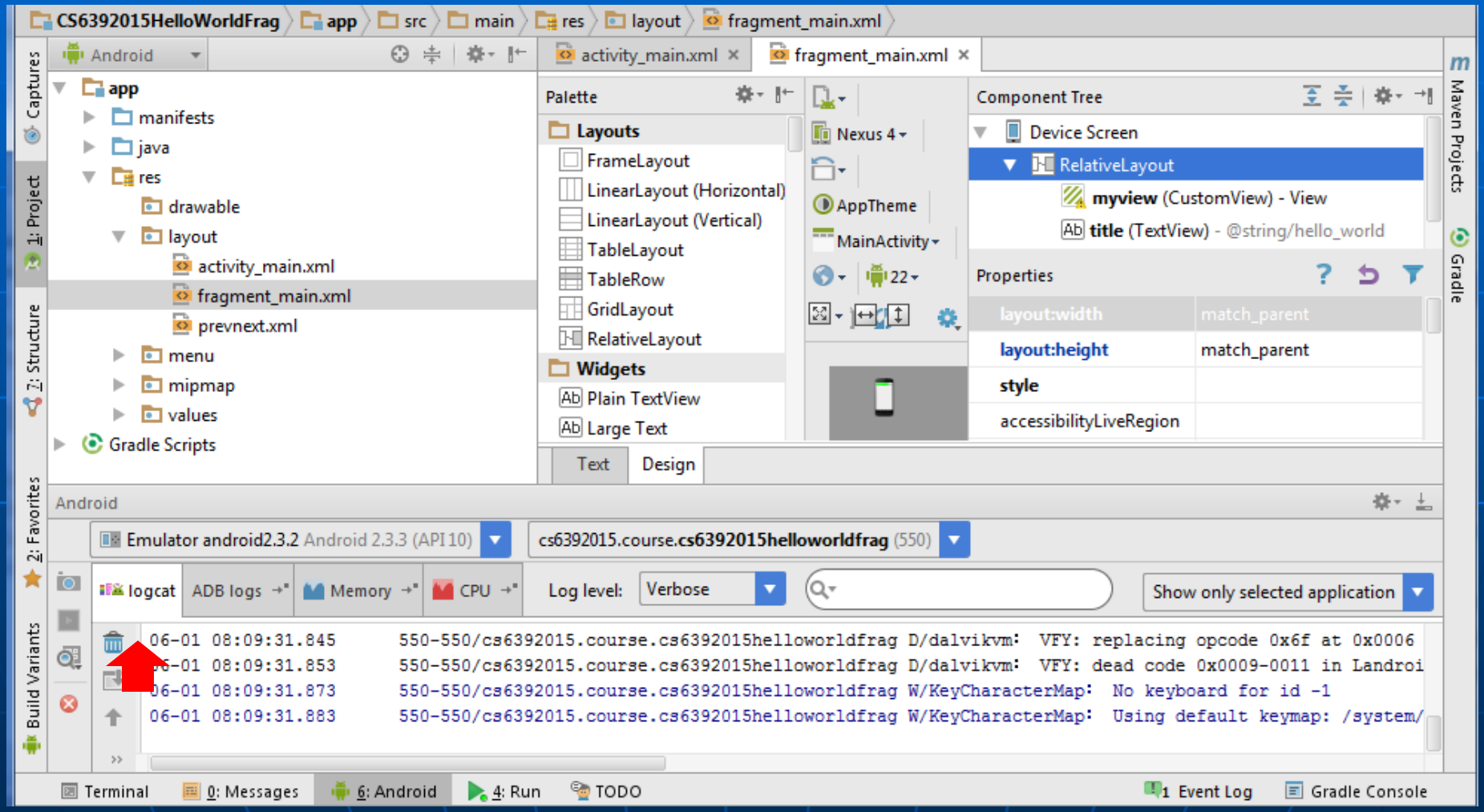
- *Log* is a class that permits to print out messages
  - v(String, String) (verbose)
  - d(String, String) (debug)
  - i(String, String) (information)
  - w(String, String) (warning)
  - e(String, String) (error)
- Example :

```
import android.util.Log;

Log.i("MyActivity", "I am here!");
```

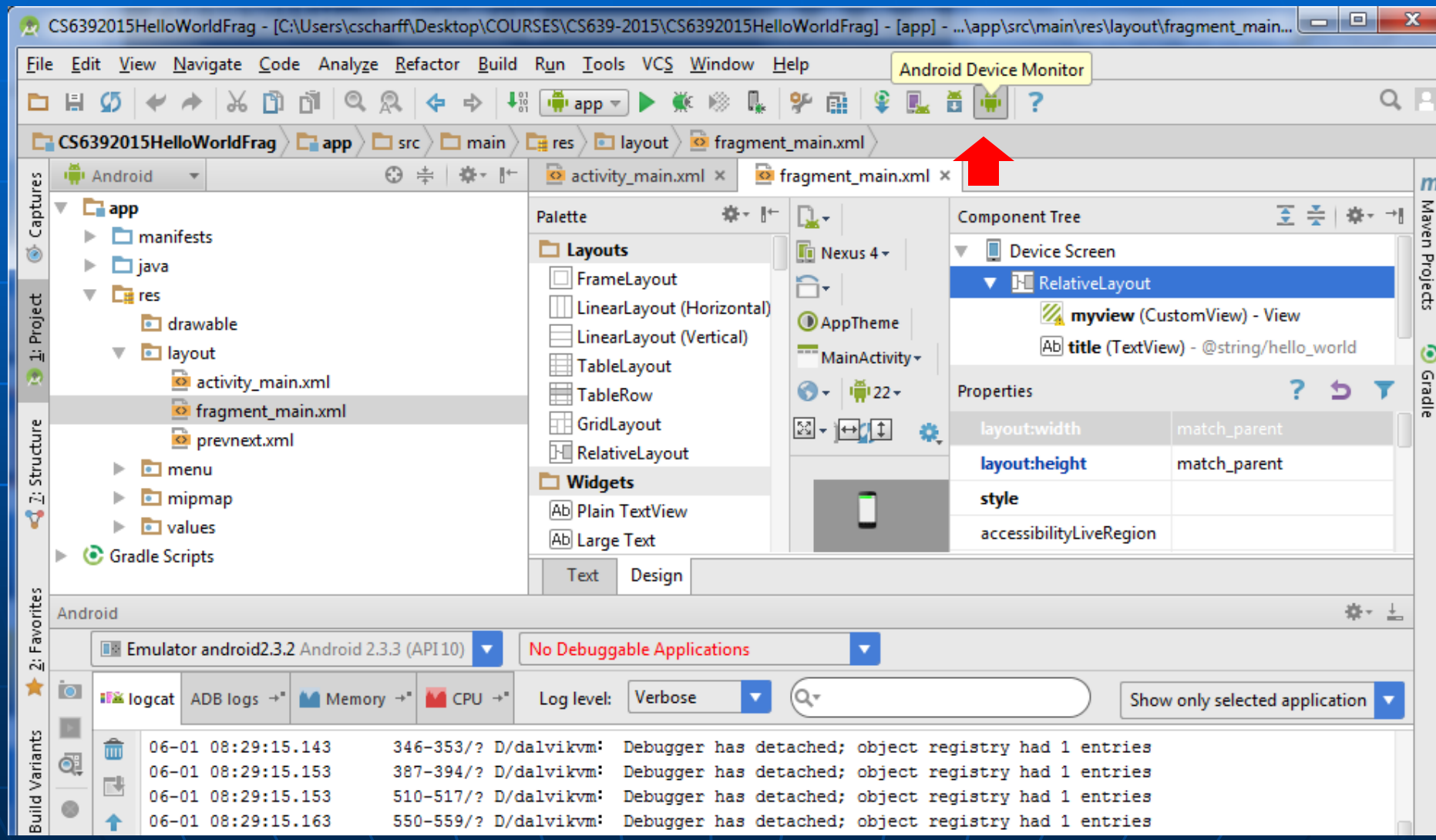
Usually the first parameter is a constant

http://developer.android.com/reference/android/util/Log.html

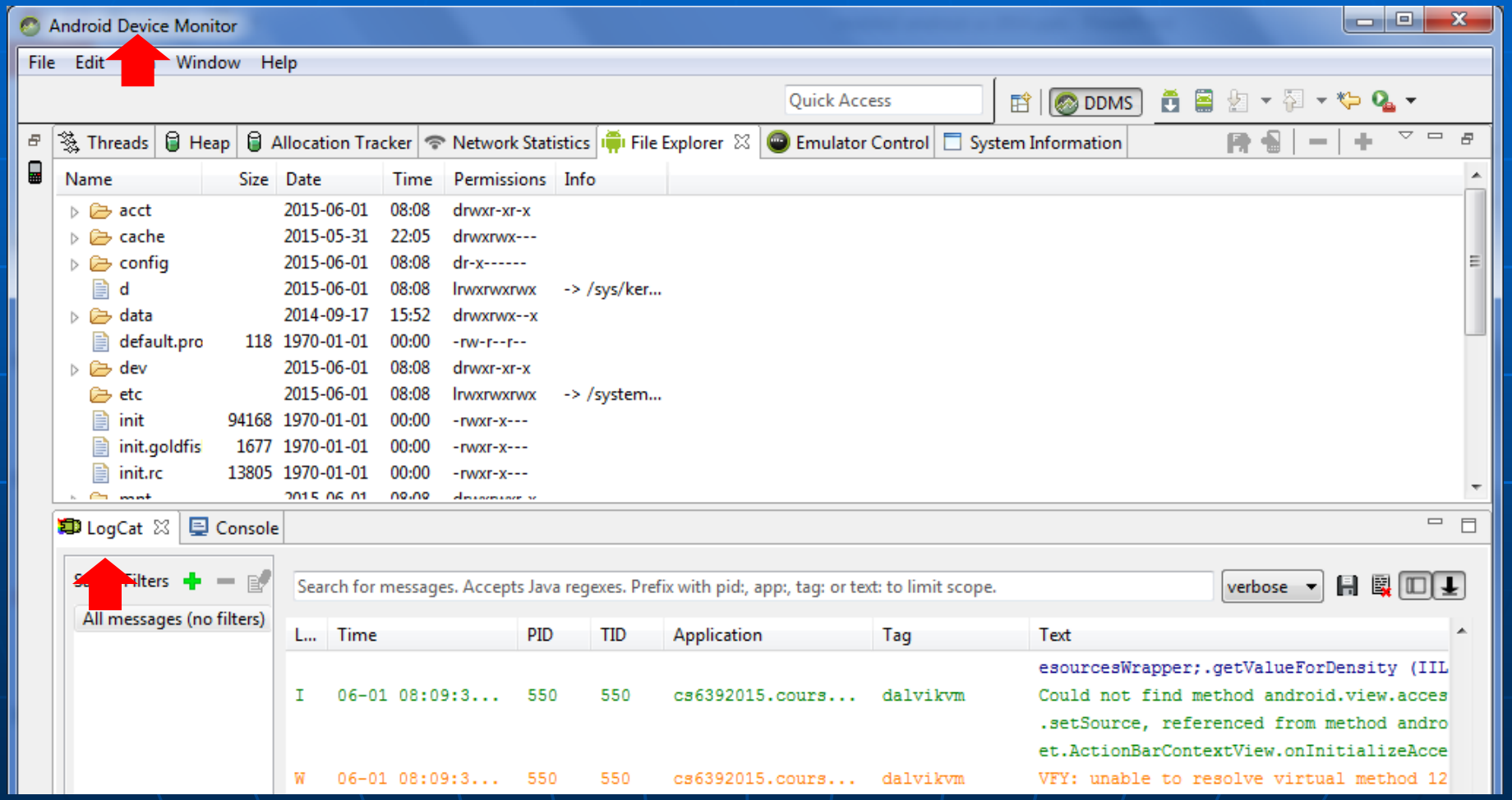
Opening Logcat in Android Studio
1st Method



Opening Logcat in Android Studio 2nd Method



Opening Logcat in Android Studio 2nd Method



Lab

- Run an app and open the LogCat