Android UI Development

Android UI Architecture

MVC in Android

Android Design Constraints

- Limited resources like memory, processing, battery
  → Android “stops” your app when not in use
- Primarily touch interaction
  → Android provides more kinds of events
- Mobile form factor
  → Android uses a “full-screen” application model
  → applications have multiple entry points that can be invoked individually
  → Dynamic layout is critical

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

Activities

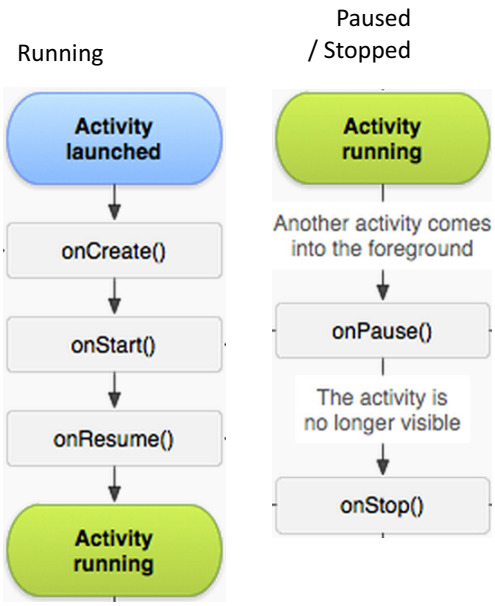
- A standard application component is an Activity
  - Typically represents a single screen
  - Main entry point (equivalent to main() method)
  - For most purposes, this is your application class

- Activity is one type of "component" we can build in Android …

Components	Description
Activity	Single-screen of an application
Service	Long-running background application
Content provider	Provides shared set of application data
Broadcast receiver	Responds to system broadcast events

Activity Lifecycle

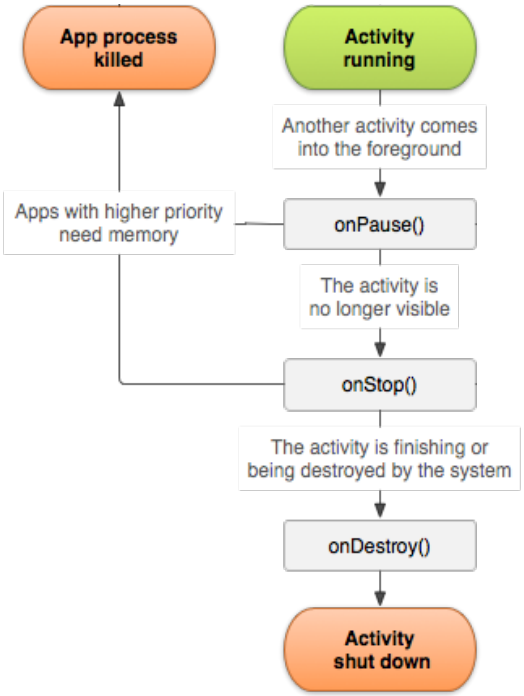
- Activities have an explicit lifecycle
  - One activity runs at a time, others are paused in the background.
  - As users navigate through your application, they switch activities.
- Every activity has a state: run, paused, or stopped.
  - Changing state fires a corresponding activity method.



https://developer.android.com/guide/components/activities/activity-lifecycle.html

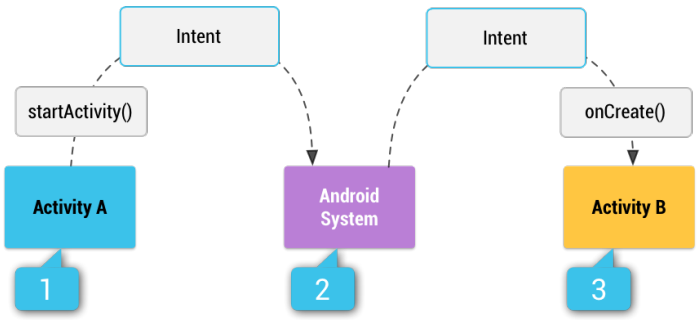
Interrupted Workflow

- Applications can stop at any time (i.e. user quits, OS kills it).
  - Each activity needs to manage its own state
  - Activities have methods for saving and restoring state



Intents

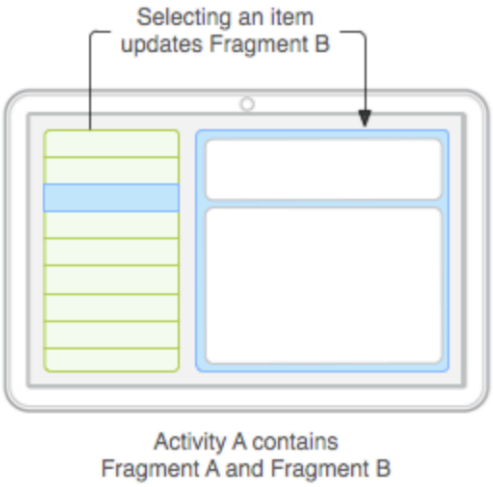
- We use **intents** to pass data between activities.
  - a data structure holding an abstract description of an action
- Use Activity startActivity() method to launch with intent.Explicit (named activity) vs. implicit (capabilities, e.g. camera)



https://developer.android.com/guide/components/intents-filters.html

Fragments

- Fragments can be thought of as portions of a UI
- An activity can contain (and manage) multiple fragments.
  - Fragments have their own lifecycle, and their own layout.
  - Alternative to multiple activities



http://developer.android.com/guide/components/fragments.html

Views (what android calls a widget)

`android.view.ViewGroup`

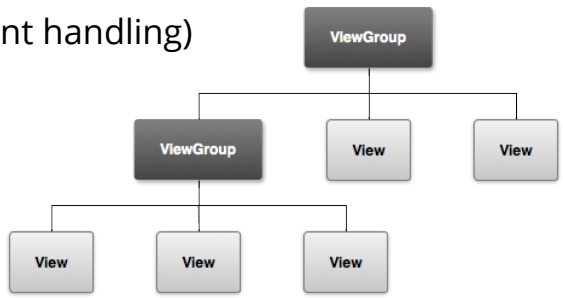
- Abstract container class
- Includes layout functionality directly
- Subclasses:

  `FrameLayout, GridLayout, LinearLayout, Toolbar, …`

`android.view.View`

- Base widget class (drawing and event handling)
- Subclasses:

  `android.widget.Button`
  `android.widget.ImageView`
  `android.widget.ProgressBar`
  `...`



UI Definition and Layout

- Layout can be handled in one of two ways:
  - **Programmatically**. You write code to instantiate ViewGroups, Views and bind them together (like in Java Swing).
  - **Use XML to describe your layout**. In XML describe the screen elements (view groups and views) along with properties, and then tell your application to dynamically load it.
- Using XML is the preferred way
  - Android Studio includes a GUI builder to make this easier!

xml layout

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingBottom="16dp"
    tools:context=".MainActivity" >

<SeekBar
    android:id="@+id/seekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
/>

<TextView
    android:id="@+id/numberField"
    android:text="X” />

</RelativeLayout>
```

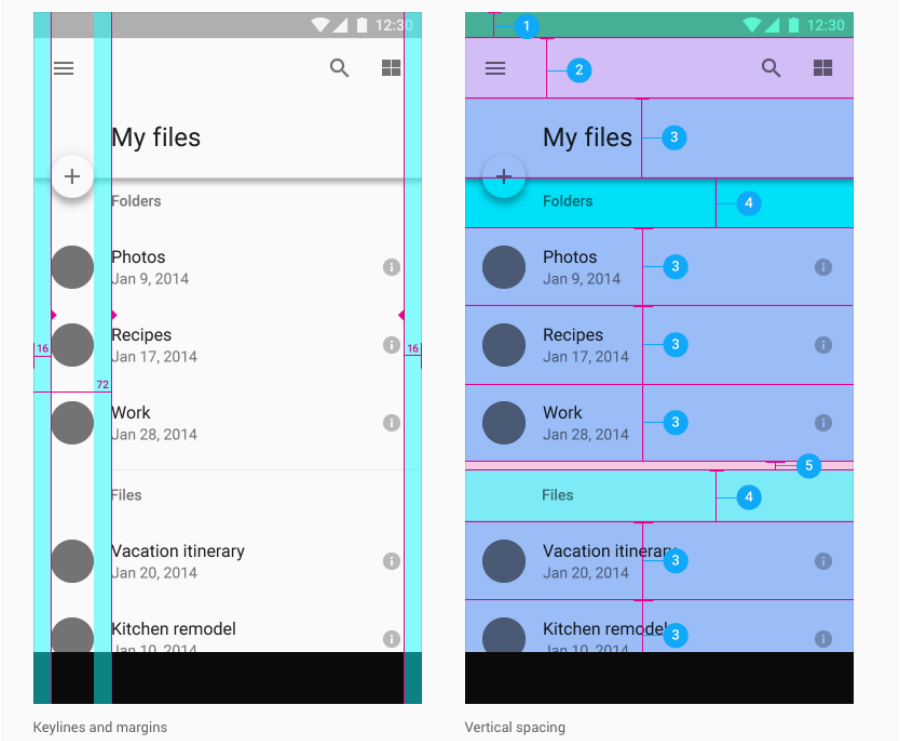
dp Layout Units

- Uses Density-independent pixels (dp), pronounced “dips”
- A dp is equal to one physical pixel on a screen with density 160
- To calculate dp:

```
dp = (width in pixels * 160) / screen density
```

Screen density	Screen width in pixels	Screen width in density-independent pixels
120	180 px	
160	240 px	240 dp
240	360 px	

Android Grid Layout System



Events

- Android uses the Java event model with additional mobile events
  - Event listener: interface for specific type of event
  - Event handler: registered callback method to handle the event

Event Listener	Event Handler	Type of event
OnClickListener()	onClick()	Touch, click
OnLongClickListener()	onLongClick()	Press and hold
onTouchListener()	onTouch()	Generic touch events; can be used for touch_up, second_touch

Building Applications

- A typical application will include:
- Activities
  - MainActivity as your entry point
  - Possibly other activities (corresponding to multiple screens)
- Views
  - Screen layouts
  - ViewGroups containing Views (i.e. components)
- Intents
  - required if you have multiple Activity screens

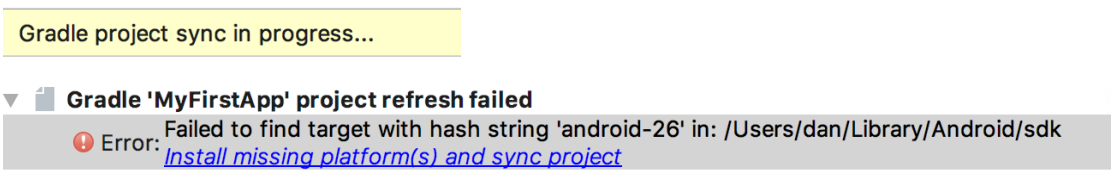
Developing Android Applications

- Android Studio
- Simple Example
- Android MVC

Android Development Environment

1. Install Android Studio, and run it
   https://developer.android.com/studio/install.html
2. Don't import previous settings
3. Choose "install standard type"

Study progress bar design while you wait …



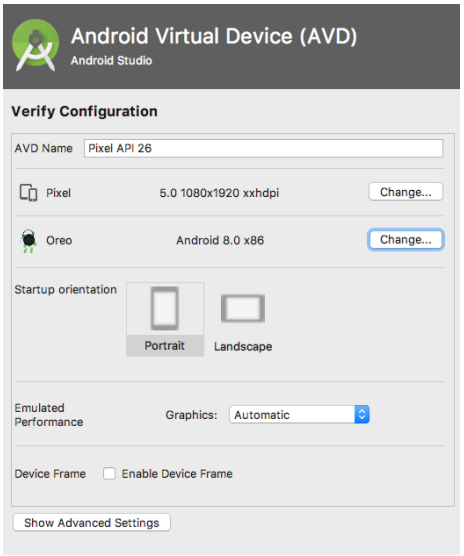
Project Components

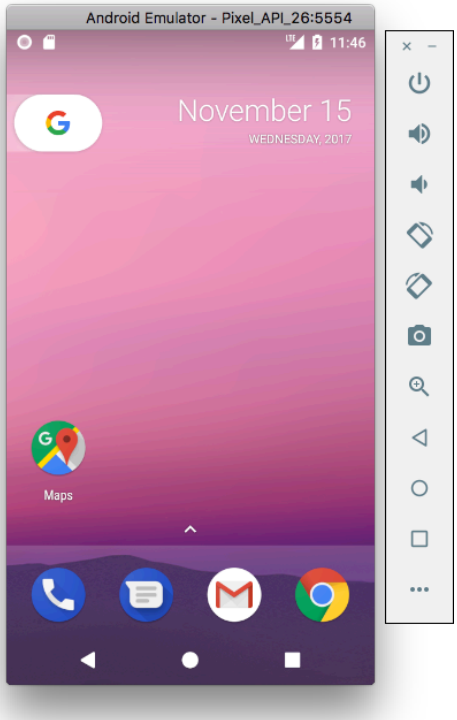
- Need to choose which API to target
  - We'll use API 15
- Company Domain
  - only important if you release your app, can just use something like: `cs349.uwaterloo.ca`

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.2%
4.2 Jelly Bean	17	96.0%
4.3 Jelly Bean	18	91.4%
4.4 KitKat	19	90.1%
5.0 Lollipop	21	71.3%
5.1 Lollipop	22	62.6%
6.0 Marshmallow	23	39.3%
7.0 Nougat	24	8.1%
7.1 Nougat	25	1.5%

Android Virtual Device (AVD)

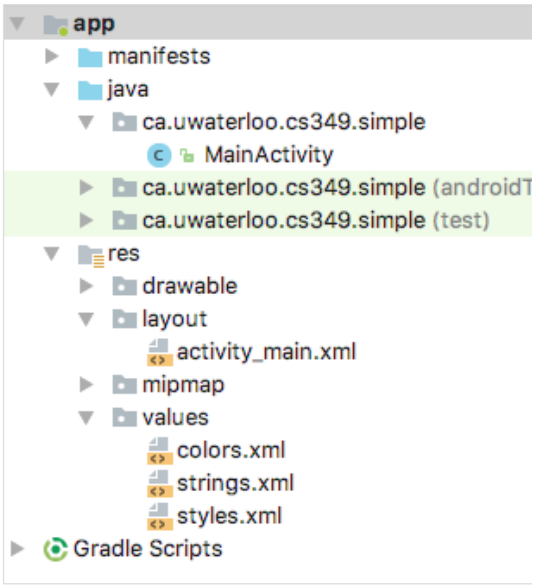
- Create an AVD for testing
  - Device: Pixel
  - System: Android 8 x86 (Oreo API 26)





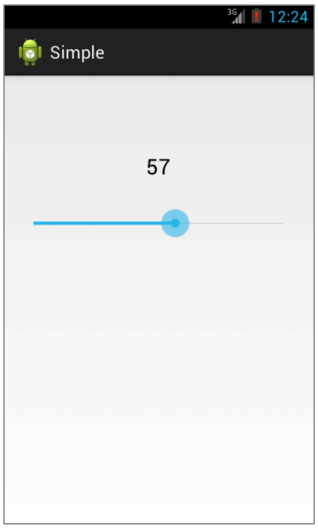
Code Demo: Simple (Android Project Files)

- Manifests
  - (AndroidManifest.xml) metadata about the app
- Java
  - **(*.java) source code**
- Resources
  - **layout: (*.xml) UI layout and View definitions**
  - **values: (*.xml) constants** like strings, colours, …
  - also bitmaps and SVG images (mipmap*, drawable*, ….)



Code Demo: Simple (Highlights)

- Notes
  - Activity
  - layouts
  - Inflating
  - accessing widgets by id
  - resources (string, colour, integer)

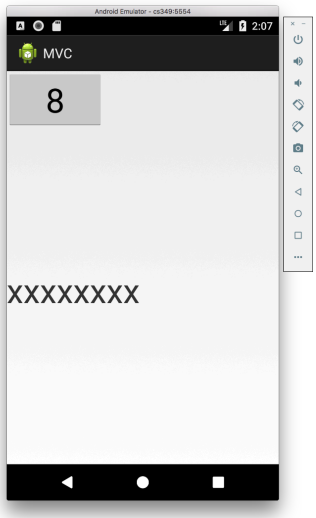


MVC in Android

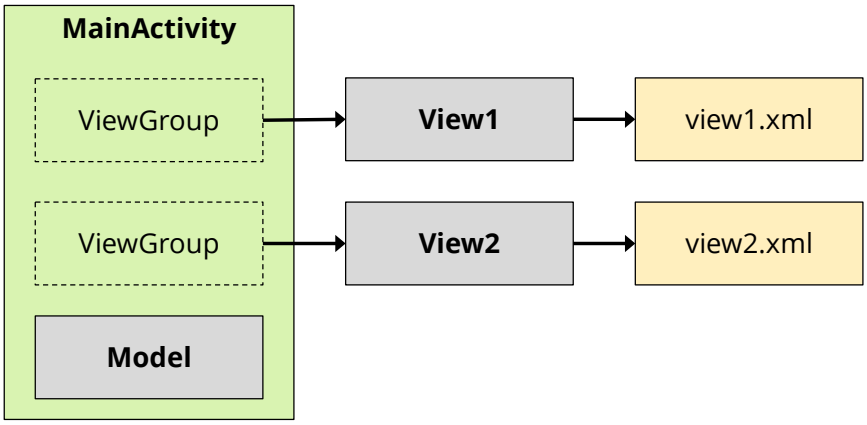
- An Android UI is XML, so there is a natural separation between the UI and application data and business logic
  - the question is how to connect the view and the model
- Google seems to recommend Model-View-ViewModel (MVVM)
  - view is connected to model through a view-model
  - the view-model manages data binding from view to model
- Other people use Model-View-Presenter (MVP)
  - where presenter is broker between view and model
- I'll show a classic MVC like we used in Java

Code Demo: MVC1

- Views as Android ViewGroups
  - similar to desktop MVC we discussed earlier
- Notes:
  - model essentially identical to desktop Java
  - "inflating" view layouts into main view
  - onPostCreate when inflating view layouts

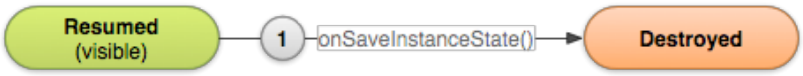


MVC1 Structure

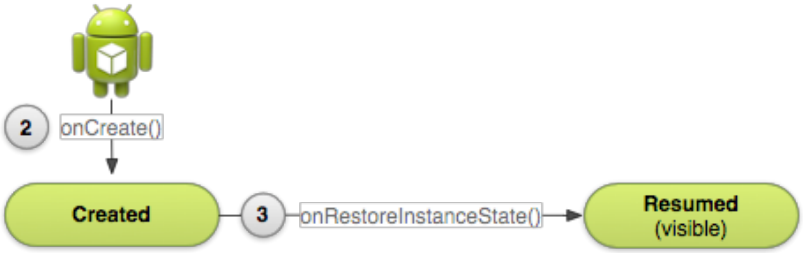


Activity UI xml Other Class

Recreating an Activity




```
protected void onSaveInstanceState(Bundle outState) {
   outState.putInt("Counter", model.getCounterValue());
   ...
}
```

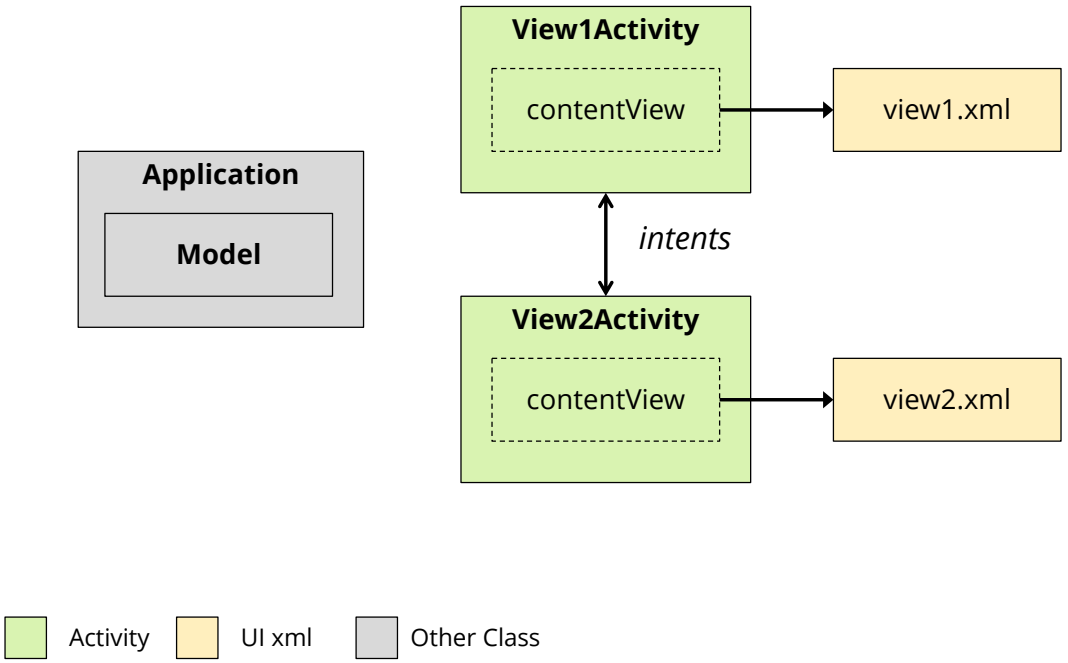



```
protected void onRestoreInstanceState(Bundle savedInstanceState) {
   model.setCounterValue(savedInstanceState.getInt("Counter"));
   ...
}
```

Code Demo: MVC2

- Views as Activities, global static model
- Notes:
  - Application class
  - onDestroy() and deleteObserver()
  - Create options menu
  - Intents to start activity
  - finish()
  - no need to persist model!
- Good pattern for many apps

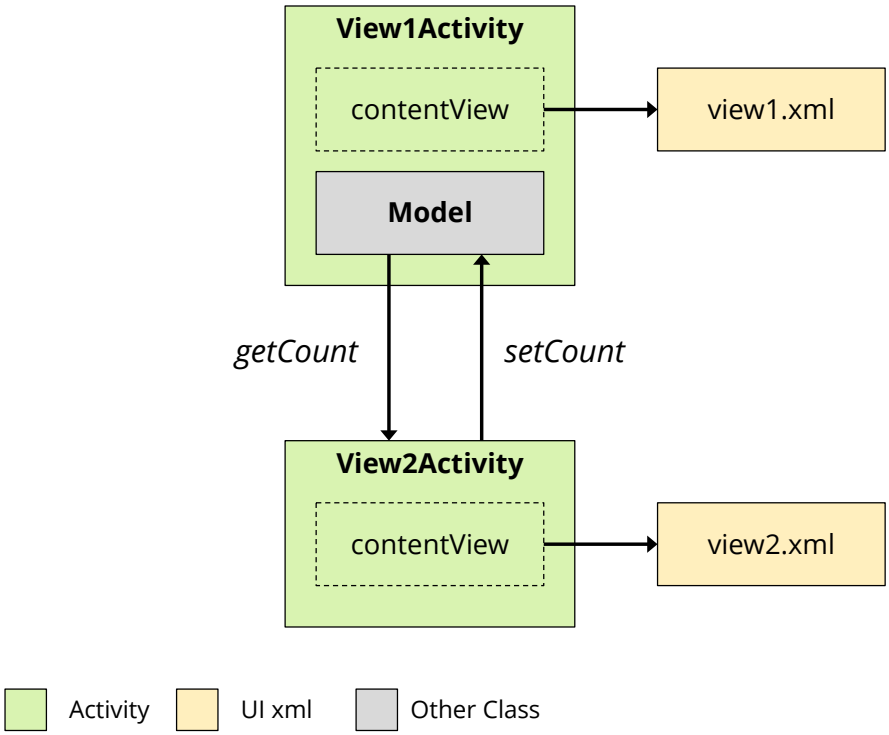
MVC2 Structure



Code Demo: MVC3

- Views as Activities, one “master” Activity
- Notes:
  - intent.putExtra and startActivityForResult
  - onResume
  - onActivityResult
  - where’s the model in second View?
  - need to save state in Master Activity
- Good for options/settings views

MVC3 Structure

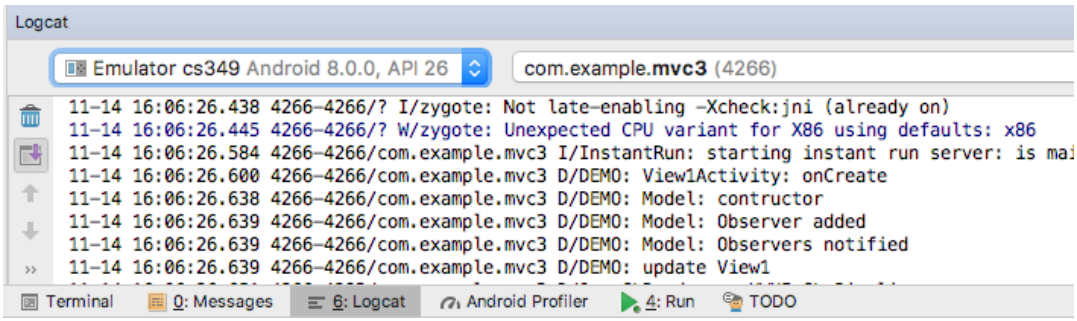


MVC in Android

- Can combine different strategies:
  - global model in Application and Master View
- Important part is to separate application data and business logic

Tips & Tricks

- AVD
  - The AVD is slow to launch, so keep it running in the background while you're programming / debugging.
- Use the debugging tools in Android Studio
  - You can set breakpoints etc. as usual
  - logcat shows device output and you can write to it using the android.util.Log class (sim. to printf).



Android MV*

- There are several MV* libraries and approaches
- “Android Architecture: MV?”
  - (also good overview of MVC variants)
  - https://doridori.github.io/Android-Architecture-MV/#sthash.TS4RmiB1.dpbs
- “How to Adopt Model View Presenter on Android”
  - https://code.tutsplus.com/series/how-to-adopt-model-view-presenter-on-android--cms-1012
- “The MVC, MVP, and MVVM Smackdown”
  - https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/