CMSC434

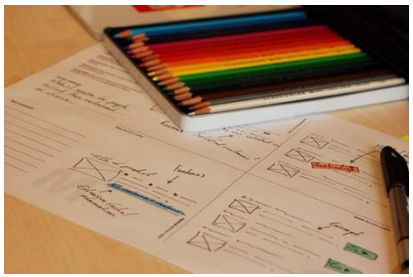
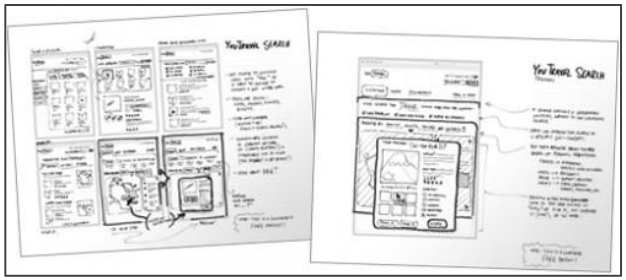
Introduction to Human-Computer Interaction

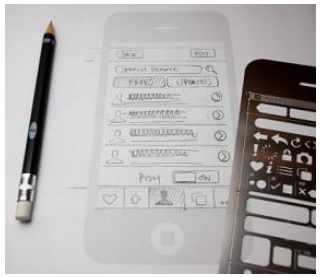
Week 08 | Lecture 16 | Oct 23, 2014

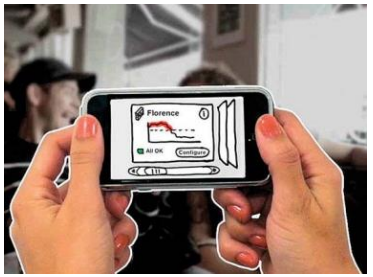
Building Android UIs II

Implementing Custom Views







Human Computer Interaction Laboratory

@jonfroehlich

Assistant Professor Computer Science

UNIVERSITY OF MARYLAND

Today

1. Logistics
2. Hall of Fame/Shame
3. Android II – Implementing a View
4. Android In-Class Activity
5. TA05 Check-In

Quiz II

I believe everyone has taken this now. Will review quizzes and post grades as well as the answers tonight

Due Thurs 10/28

TA05 Paper Prototypes & Testing

Edit 5 ...

Paper Prototypes & User Testing

Posted: Friday, October 10
Deadline: Tuesday, October 28 (before classtime)
Point Total: This assignment is worth approximately 11% of your project grade (4.5% overall).

Assignment Overview

Your assignment is to convert a subset of your more promising sketches from the last assignment into paper prototypes and to test them with four users. You will create and **test two different versions** of your prototypes. As I've noted before in class, comparing two designs is far easier/better than evaluating one single design in isolation. For example, it's easier for a user to express preference between one design over another rather than articulating, exactly, why they don't like a single design.

Following from the IRB protocols we discussed in class, your participants must sign an informed consent document before taking part in your test. With the participant's permission, you should **video record all sessions**. This can be done with any type of recording device from a DSLR to a mobile phone camera.

Your report should include figures of your paper prototypes, a description of your study method, and a breakdown of your primary findings from the user testing. The video recordings will be used later in the semester for TA10 (so do not lose them).

What to Do

1. **To start, watch the Nielsen Norman** Paper Prototyping: A How-To Video. This will help prepare you for this project assignment and the next one as well. In addition, prototyping--and, specifically, rapid prototyping--is perhaps one of the most valuable techniques in HCI/design. So, it's worth spending some time on the readings and, of course, applying the prototyping concepts in your projects.

2. **Iterate and refine the three primary tasks** that users should be able to accomplish with your application (based on learnings/reflections from the last assignment).

3. **Then, transition to the first core part of the assignment: riff, iterate, and create two different paper prototypes for the three primary tasks.** That is, you must create "Paper Prototype #1" that allows your users to accomplish the three tasks one way and "Paper Prototype #2" that allows your users to accomplish the three primary tasks another way. The paper prototypes should be functionally different so that your users can compare and contrast their experiences with both of them. So, for example, the two prototypes should represent the tasks in fundamentally different ways. Remember, the focus here is not on aesthetics/beauty but rather on understandability, usability, approachability, and, to some degree, layout, widget type, etc.

4. **Once, you've created the two paper prototype designs, beta test them with members of your team and make requisite changes.** This "eating your own dog food" is a good way to catch errors before investing time in testing with actual users.

5. **Now you're ready for real user testing (the second core part of the assignment). Recruit four independent users to test out your paper prototypes.** These users cannot be members of the class; however, they can be members of Dr. Vibha Sazawal's section. Each user testing session must be done in isolation (that is, you cannot have more than one user testing at a time) with at least two experimenters present. For the testing session, you should follow this protocol:

    1. First, download and modify this IRB "informed consent" template to fit your project [link]. At the beginning of the user testing session, read the "Purpose of this Study" section of the consent form out loud to your participants. This should be done consistently for each participant. Then, give your participants a chance to read the entire consent form, ask questions, and, if they agree to participate, have them sign the form. If they do not agree to participate, simply wish them a nice day and recruit another participant (it can be slightly awkward but this happens!). If they do agree to participate, provide a copy of the form and take the signed copy for yourself (please scan in the signed consent form and include it in your report appendix).

    2. After the informed consent process, you can begin user testing the prototypes. Follow the method described in the Nielsen Norman "Paper Prototyping: A How-To Video." Because you have two different prototype designs for your tasks, you should fully test one

Wiki Navigation
Projects +
Recent Changes
Pages and Files +
Members +
Settings

Course Pages
Home
Syllabus
Lectures
Readings

Individual Assignments
Hall of Fame/Shame
IA01 Background Survey - 9/3
IA02 Project Pitch - 9/7
IA03 Pitch Vote - 9/9
IA04 Understanding Metro Users - 10/2
IA05 Building an Android UI

Team Project Assignments
TA01 Group Collaboration Plan - 9/16
TA02 Formative Research - 9/28
TA03 Proposal Presentation - 9/30
TA04 Sketches & Storyboards - 10/14
TA05 Paper Prototypes - 10/28

Project Peer Assessment
Peer Assessment (TA01-TA03) - 10/7

Reading Assignments
R01 Brainstorming - 9/4
R02 Task-Centered Design - 9/9
R03 IDEO Shopping Cart - 9/11
R04 Ethnographic Approach - 9/16
R05 Prototyping - 9/23
R06 Presentation Tips - 9/30
R07 Sketching & Storyboards - 10/7
R08 Principles of Interaction - 10/14
R09 Cog Sci & Design - 10/21

edit navigation

Due Tues 11/11
Work with Partner / Start immediately!

IA05 Building an Android UI

Edit | 5 | ...

Building an Android UI

Posted: Monday, October 20
Due: Tuesday, November 11, 7AM
Point Total: This assignment is worth approximately 5% of your overall grade.
This assignment is to be completed with one CMSC434 partner of your choice.

Assignment Overview

In this assignment, you will design and implement a custom clock application for Android using two common programmatic approaches: first, using the Android Studio built-in design toolkit (Figure 1) to select, drag-and-drop, and layout widgets visually (along with XML); second, using a custom view that overrides the onDraw() method (Figure 2). Both clock implementations must show and update the current time (in real-time). While the WYSIWYG visual editor is convenient, relatively easy-to-understand, and useful for a wide range of tasks, the built-in widgets are rather limited. Most applications combine off-the-shelf widgets (*e.g.*, textboxes, dropdown menus, buttons) with their own views.

Android Graphical UI Builder

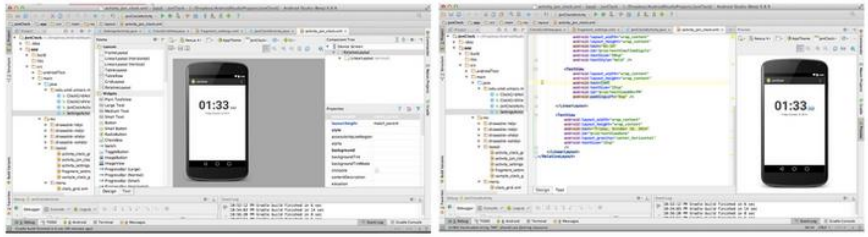


Figure 1. The Android graphical UI builder (WYSIWYG editor) allows you to build your interfaces visually—via a drag-and-drop interface—paired with an XML-based declarative syntax.

Android Custom Views

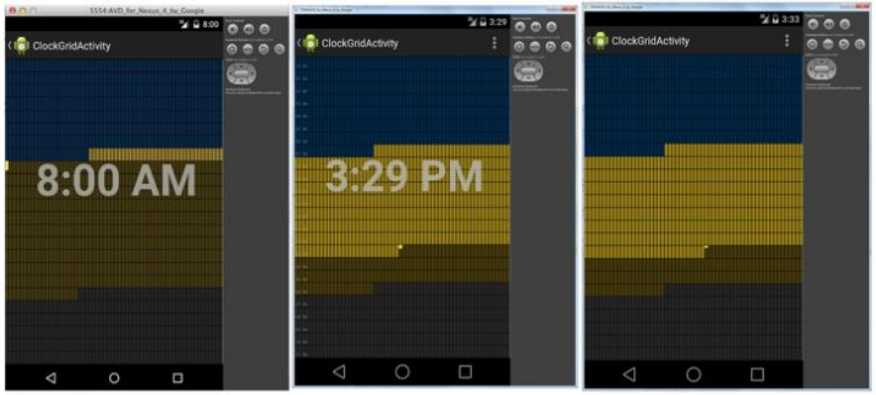


Figure 2. Many applications on your smartphone are built using custom views where the

Projects
Recent Changes
Pages and Files
Members
Settings

Course Pages
Home
Syllabus
Lectures
Readings

Individual Assignments
Hall of Fame/Shame
IA01 Background Survey - 9/3
IA02 Project Pitch - 9/7
IA03 Pitch Vote - 9/9
IA04 Understanding Metro Users - 10/2
IA05 Building an Android UI

Team Project Assignments
TA01 Group Collaboration Plan - 9/16
TA02 Formative Research - 9/28
TA03 Proposal Presentation - 9/30
TA04 Sketches & Storyboards - 10/14
TA05 Paper Prototypes - 10/28

Project Peer Assessment
Peer Assessment (TA01-TA03) - 10/7

Reading Assignments
R01 Brainstorming - 9/4
R02 Task-Centered Design - 9/9
R03 IDEO Shopping Cart - 9/11
R04 Ethnographic Approach - 9/16
R05 Prototyping - 9/23
R06 Presentation Tips - 9/30
R07 Sketching & Storyboards - 10/7
R08 Principles of Interaction - 10/14
R09 Cog Sci & Design - 10/21

edit navigation

Published

Post Your Android UI Partner Below!

Jon Froehlich

Oct 2 at 1:55pm

Please post your Android UI partner below.

Search entries or author | Unread | Subscribed

Reply

Alexander Whipp
Yesterday
Alexander Whipp and Enoch Hsiao

Reply

Omair Javed
Yesterday
Omair Javed and Peter Karp

Deleted by Ian Kahn on Oct 22 at 9:25pm

Reply

Ian Kahn
Yesterday
John Purtilo and Ian Kahn

Reply

Due Oct 30, 7AM
No Reading Response Required

R10 Android UI

Edit | 4

Posted: Thursday, October 23
Due: Thursday, October 30, 7AM

Android Programming and UI Building Readings

1. **Required:** A developer.android.com two-part tutorial on Building a Simple User Interface and Starting Another Activity. We essentially did part one in class on Tuesday, Oct 21st. You can download the code here and watch a video about how to load and run it in Android Studio. We did not do the 2nd part (Starting Another Activity), however, and you will need to do this for IA05 (to be able to launch the settings screen and any secondary views that you have).

2. **Required:** developer.android.com, UI Overview. The User Interface section of the Android Developer Guide is quite large, so I just want you to read the UI Overview (a few paragraphs) and then peruse/explore the other subsections. I encourage you to look at Layouts, Input Controls, Styles and Themes, and Settings as this will be helpful for IA05. However, the best way to learn about these things is to dive in and try them in Android Studio!

3. **Optional:** developer.android.com, Application Fundamentals [source link]. We do not have time in this class to go over Android programming in general but I encourage you to read this page because it explains the core aspects of Android including Activities and Intents (both of which you will use in IA05).

No Reading Response

There is no reading response for this assignment; however, content from the readings will be on the midterm. Specifically, for the exam, I expect you to have an understanding of the Android UI builder, the common Android widgets, a basic understanding of the XML structure (*e.g.*, how to specify layouts, sizes, hook up events), how to implement your own View (override the onDraw method, draw lines, shapes, and text to screen), and a basic understanding of why Android discourages absolute pixels to define distances/sizes (link).

Wiki Home
Pages
Recent Changes
Pages and Files
Members
Settings

Course Pages
Home
Syllabus
Lectures
Readings

Individual Assignments
Hall of Fame/Shame
IA01 Background Survey - 9/3
IA02 Project Pitch - 9/7
IA03 Pitch Vote - 9/9
IA04 Understanding Metro Users - 10/2
IA05 Building an Android UI - 11/11

Team Project Assignments
TA01 Group Collaboration Plan - 9/16
TA02 Formative Research - 9/28
TA03 Proposal Presentation - 9/30
TA04 Sketches & Storyboards - 10/14
TA05 Paper Prototypes - 10/28

Project Peer Assessment
Peer Assessment (TA01-TA03) - 10/7

Reading Assignments
R01 Brainstorming - 9/4
R02 Task-Centered Design - 9/9
R03 IDEO Shopping Cart - 9/11
R04 Ethnographic Approach - 9/16
R05 Prototyping - 9/23
R06 Presentation Tips - 9/30
R07 Sketching & Storyboards - 10/7
R08 Principles of Interaction - 10/14
R09 Cog Sci & Design - 10/21
R10 Android UI - 10/30

edit navigation

Hall of Fame

Hall of Shame

Light Switch Mapping/Feedback

Submitted by CMSC434 Student Enoch H.

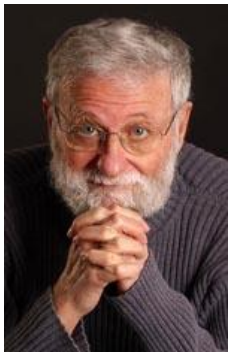
Light Switch Mapping/Feedback

Submitted by CMSC434 Student Enoch H.

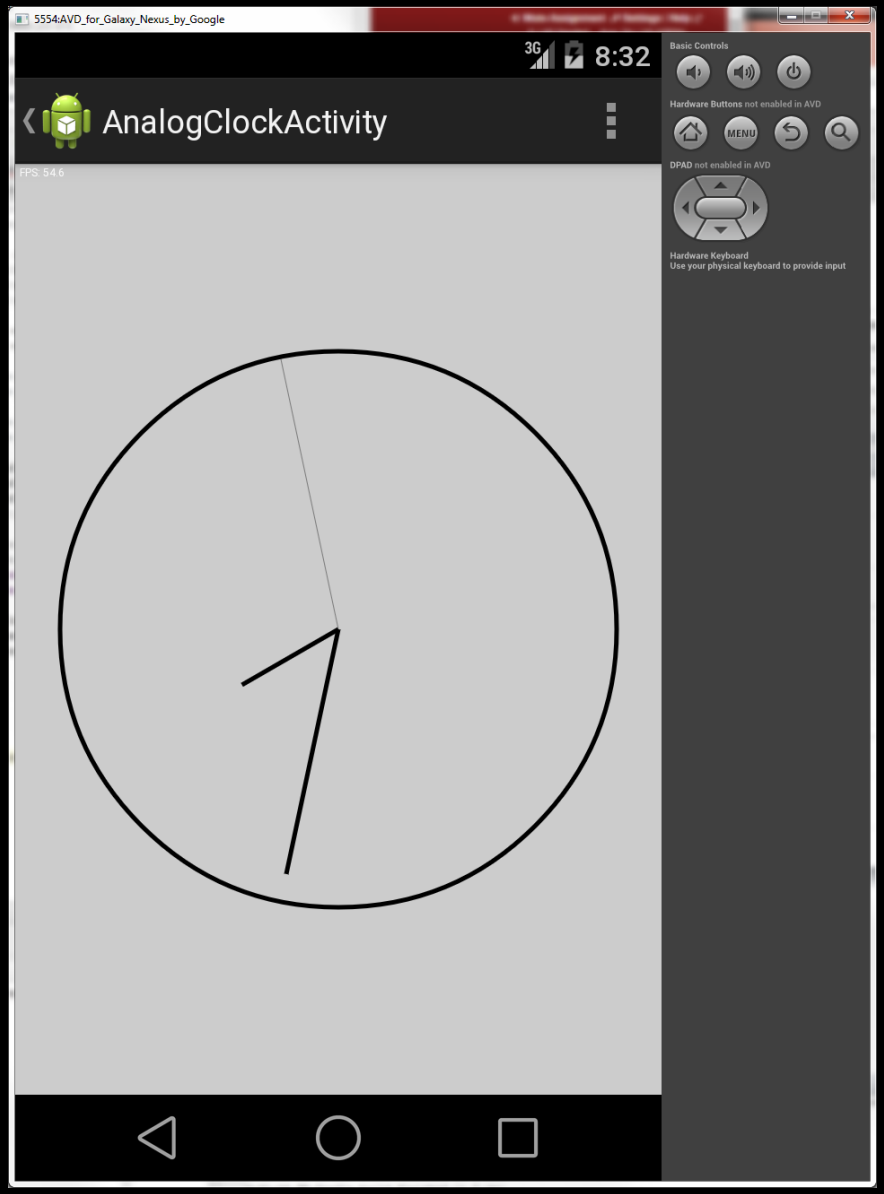
FEEDBACK

Feedback **communicates** the **results of an action**; it's a way of letting you know that the system is working on your request. Feedback must be immediate—even a delay of a tenth of a second can be disconcerting.

Don Norman
Author/Design Critic
From: The Design of Everyday Things, 2013, p.23 - 25

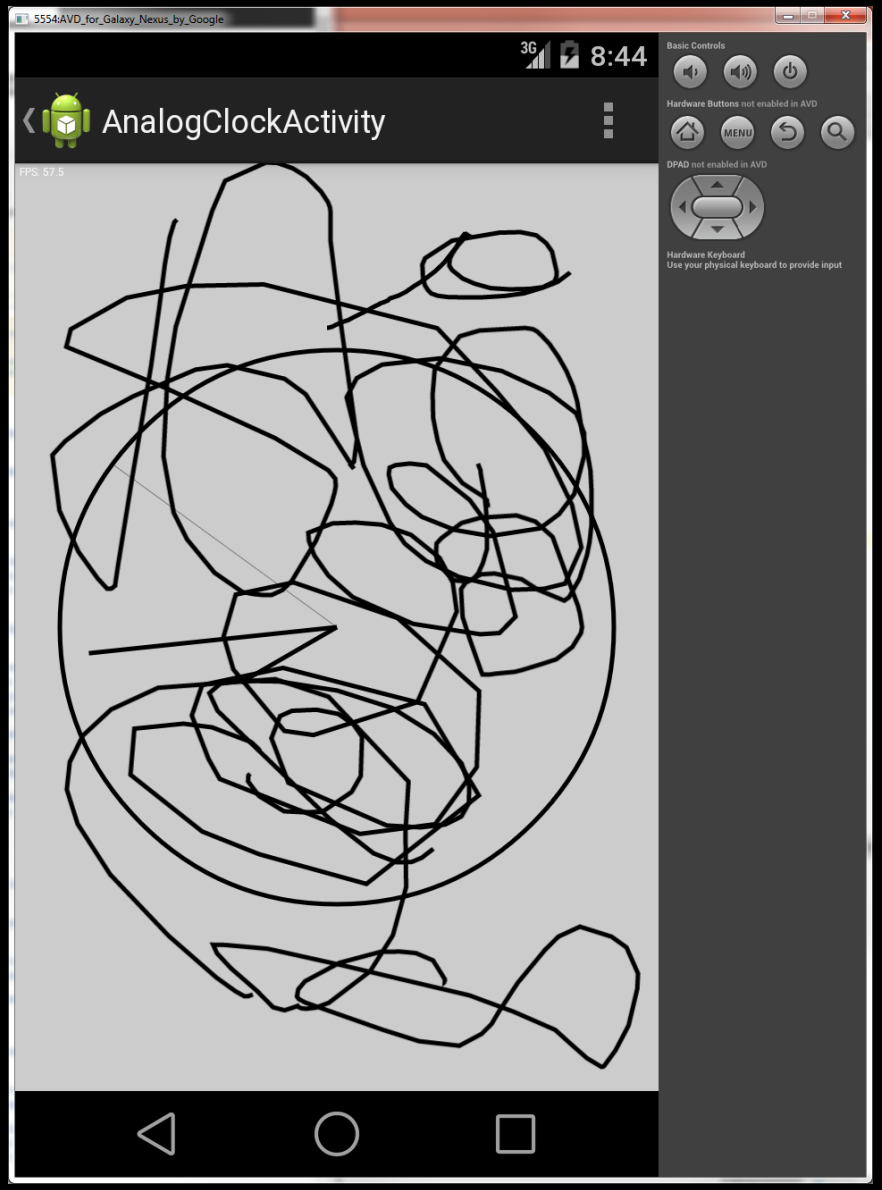


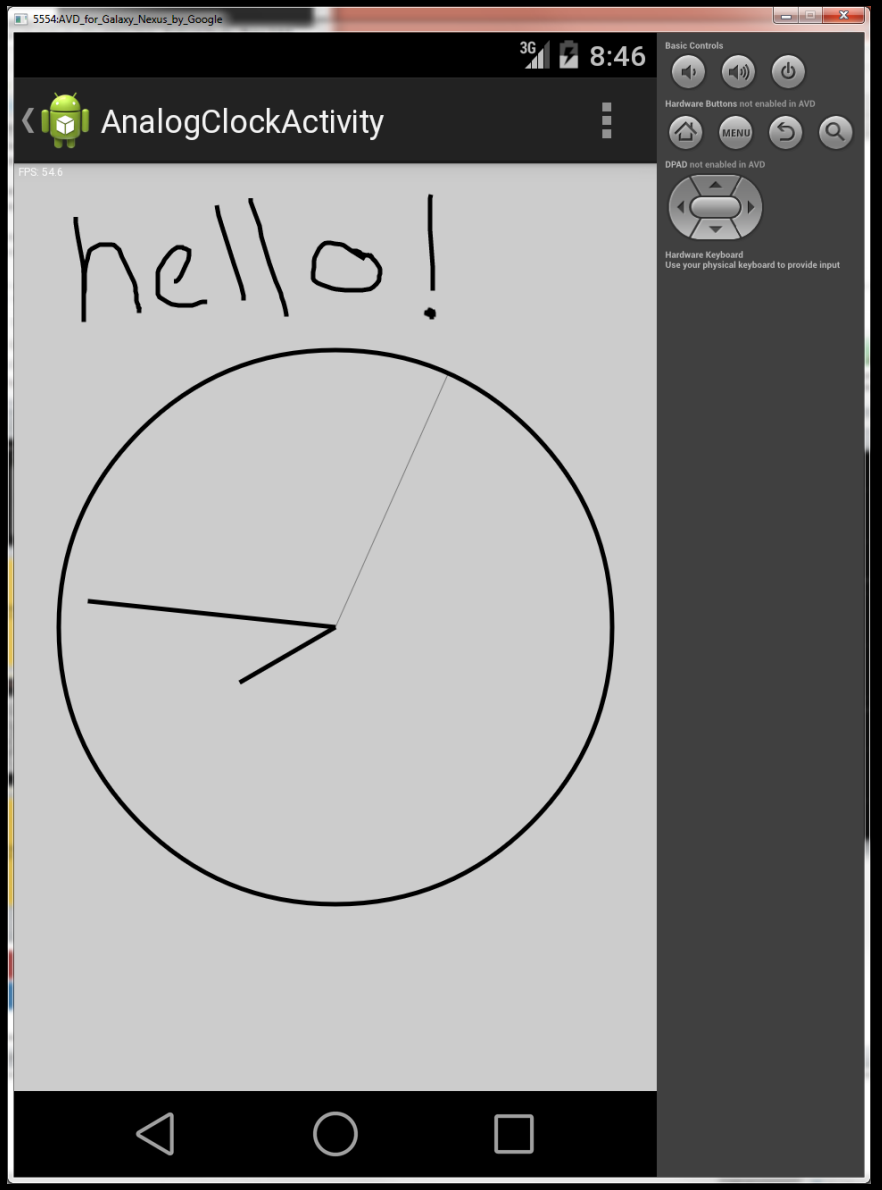
Implementing a Custom View in Android

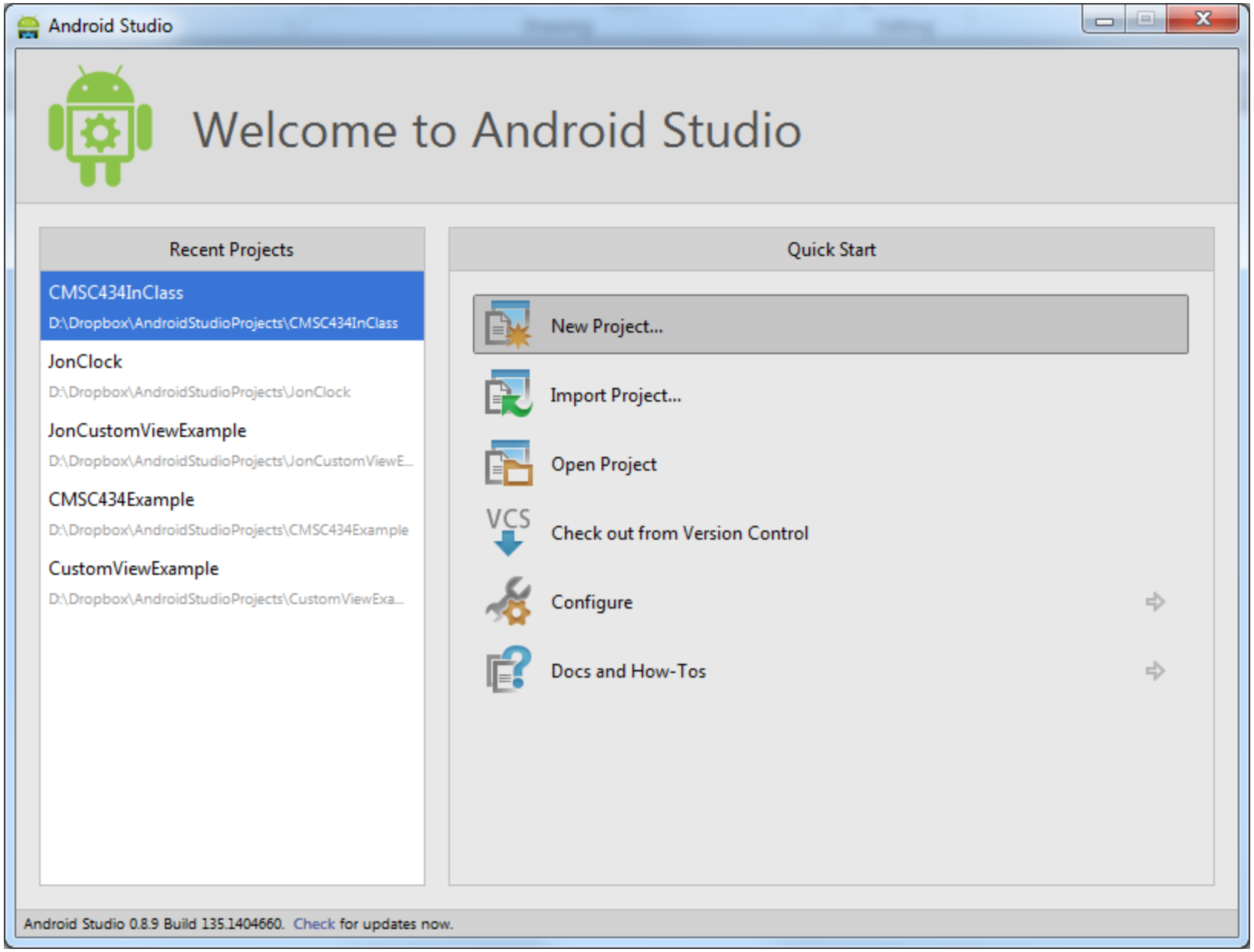


Android In-Class Activity

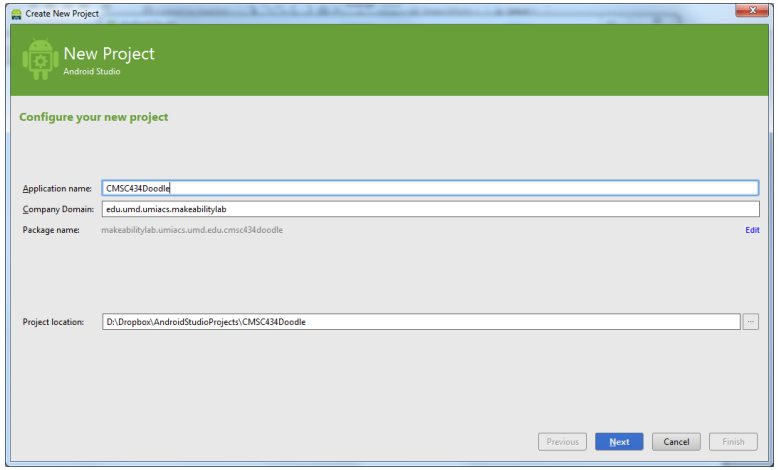
Write a quick doodle app


5554:AVD_for_Galaxy_Nexus_by_Google
8:44
AnalogClockActivity
FPS: 57.5
Basic Controls
Hardware Buttons not enabled in AVD
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

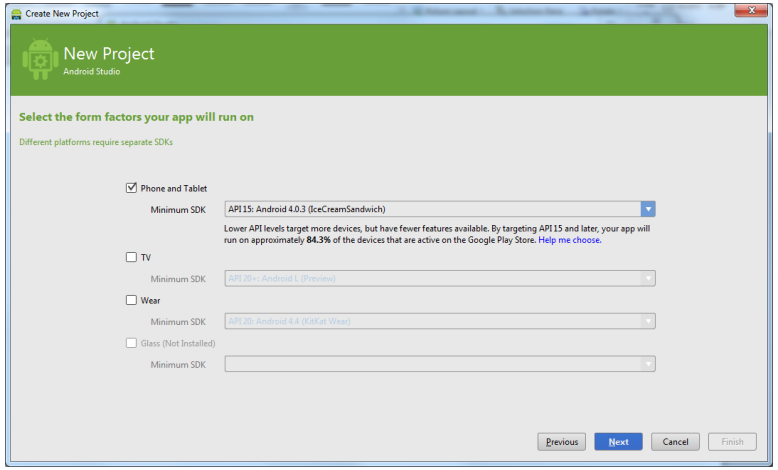

5554:AVD_for_Galaxy_Nexus_by_Google
8:46
AnalogClockActivity
FPS: 54.6
hello!
Basic Controls
Hardware Buttons not enabled in AVD
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input


Android Studio
Welcome to Android Studio
Recent Projects
CMSC434InClass
D:\Dropbox\AndroidStudioProjects\CMSC434InClass
JonClock
D:\Dropbox\AndroidStudioProjects\JonClock
JonCustomViewExample
D:\Dropbox\AndroidStudioProjects\JonCustomViewE...
CMSC434Example
D:\Dropbox\AndroidStudioProjects\CMSC434Example
CustomViewExample
D:\Dropbox\AndroidStudioProjects\CustomViewExa...
Quick Start
New Project...
Import Project...
Open Project
VCS
Check out from Version Control
Configure
Docs and How-Tos
Android Studio 0.8.9 Build 135.1404660. Check for updates now.

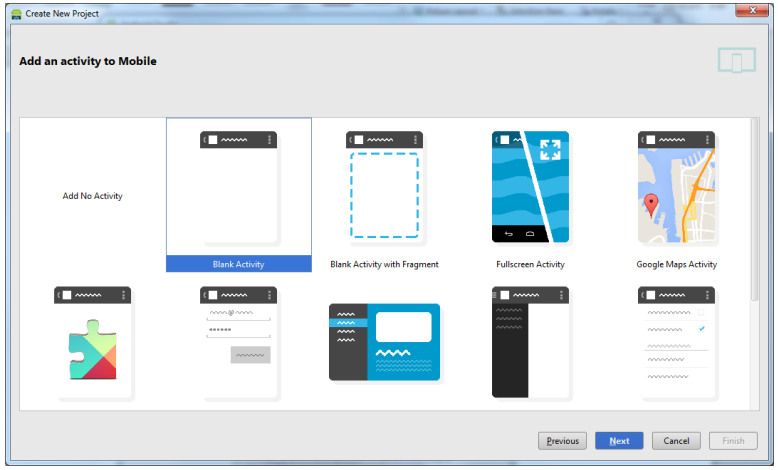
Create New Project



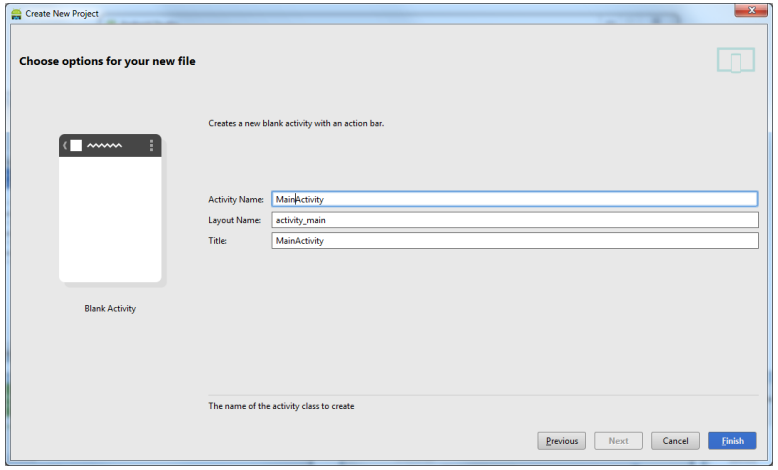
Name your application. In this case, we'll use "CMSC434Doodle" and we'll set the company domain to my research group (but again, you can use whatever you want)



The default is API15, so let's just use that.

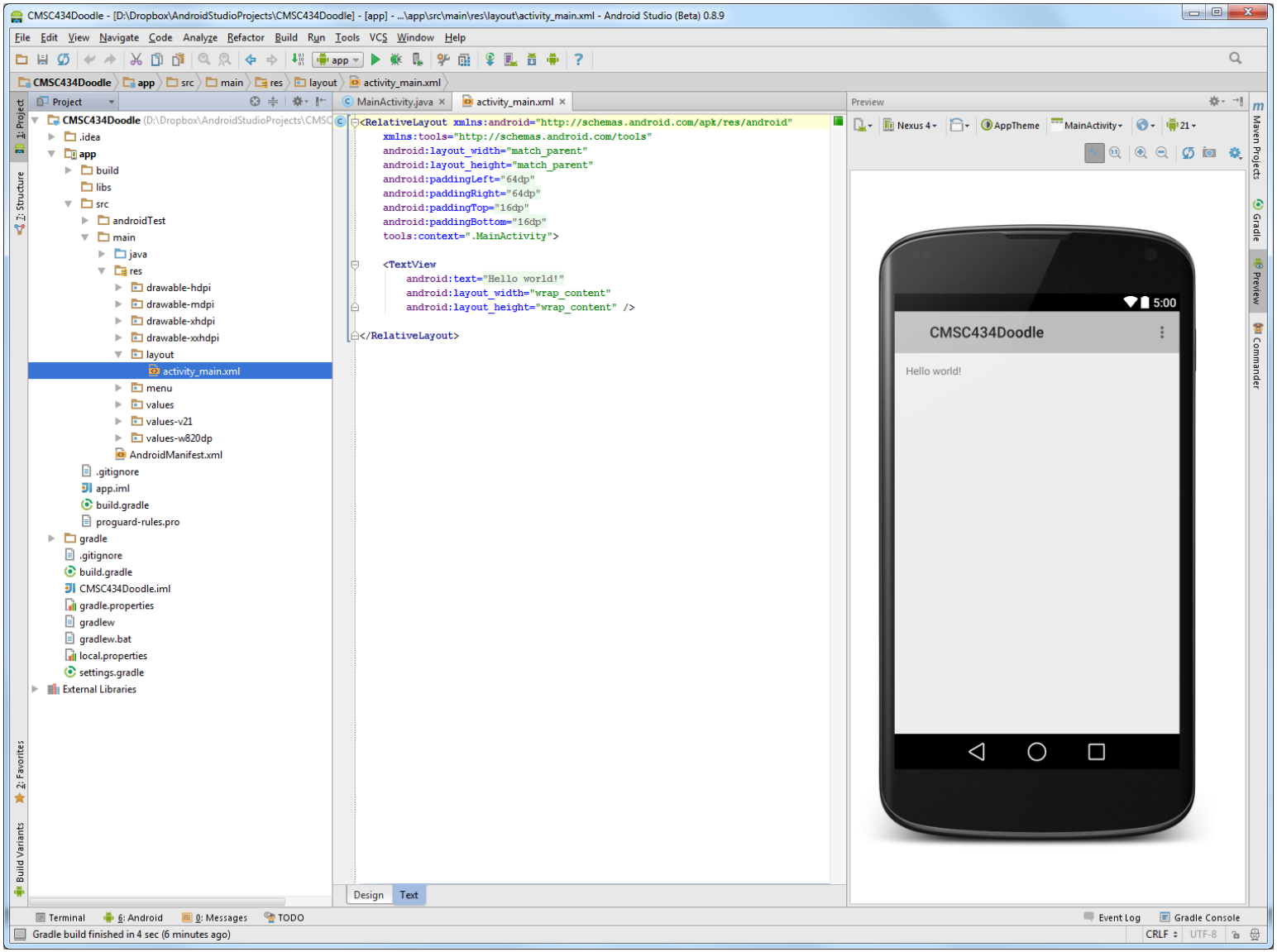


We'll use a Blank Activity. To learn more about different activity types, read the Using Code Templates guide.



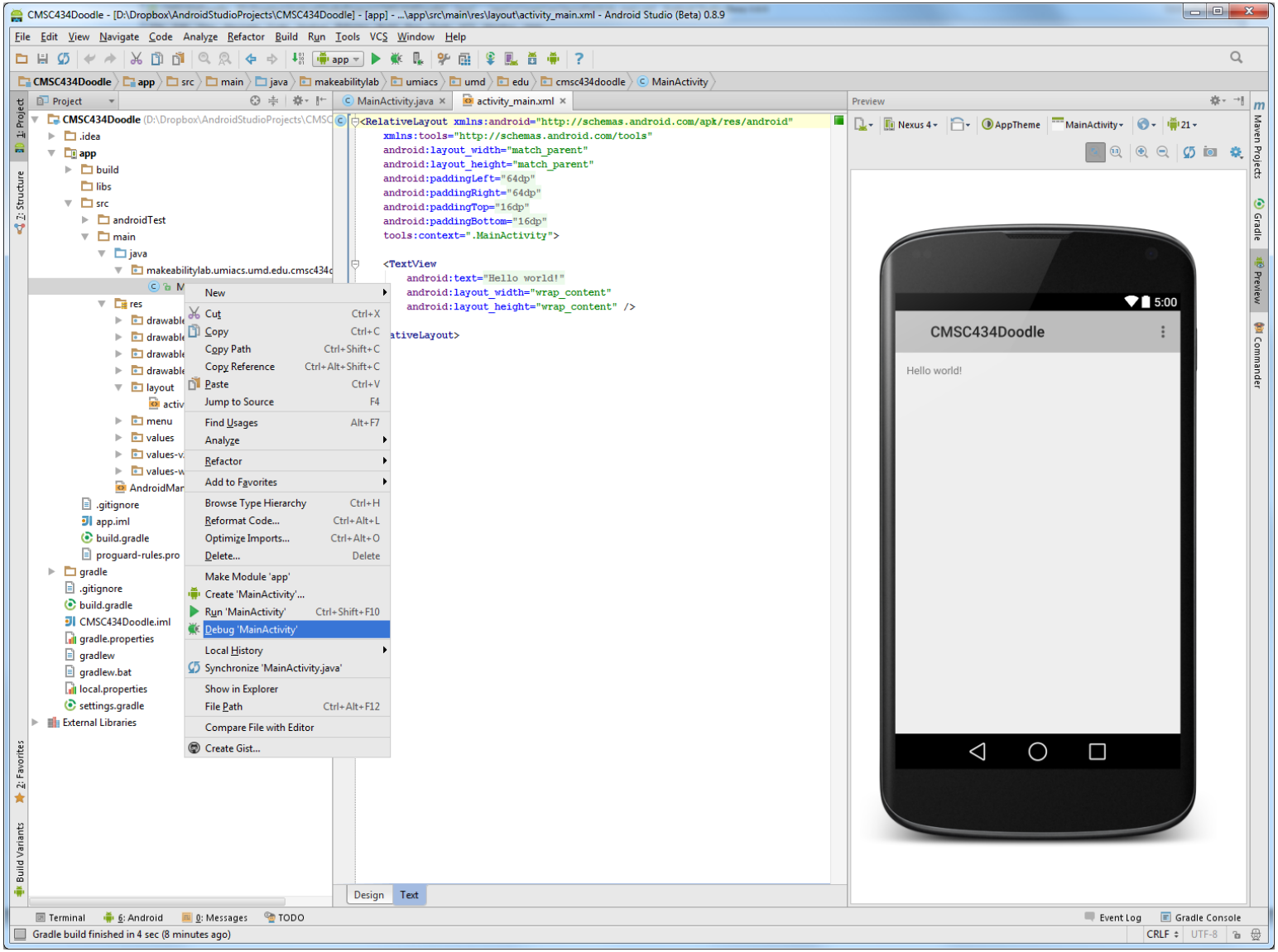
Rename the activity name to MainActivity (convention). The XML file will auto-rename accordingly.

New Project With Auto-Generated Code



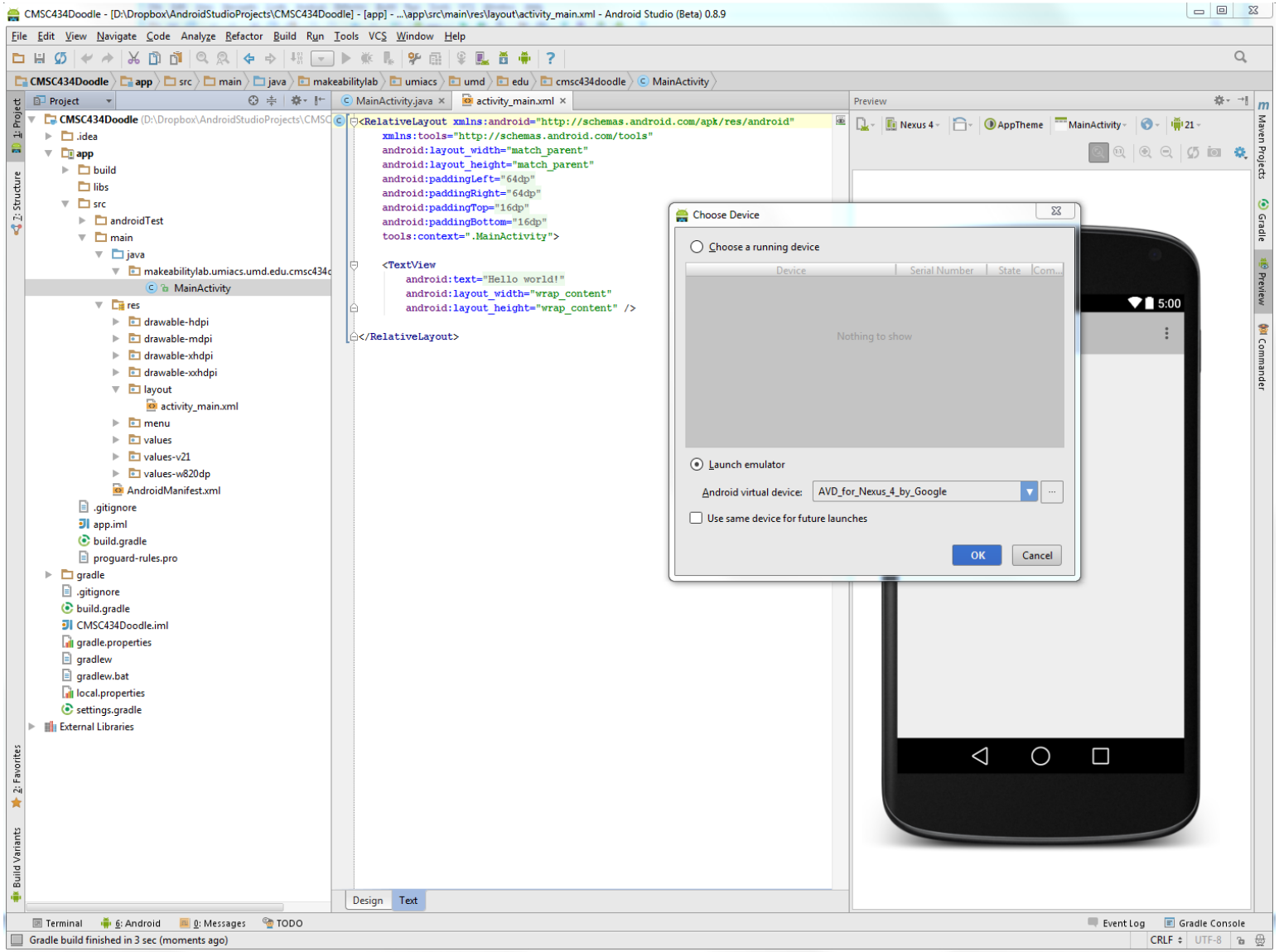
The first thing I like to do with a new project, is compile and run. So, let's do that.

New Project With Auto-Generated Code



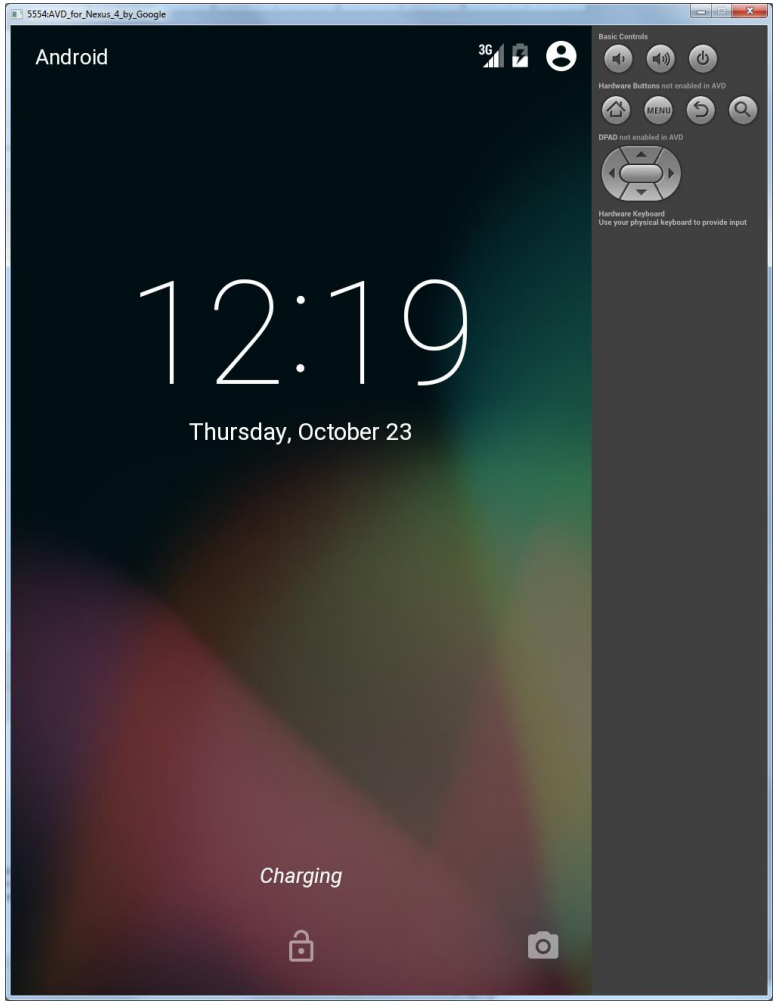
Select MainActivity in the java folder, right-click, and click "Debug 'MainActivity'"

New Project With Auto-Generated Code

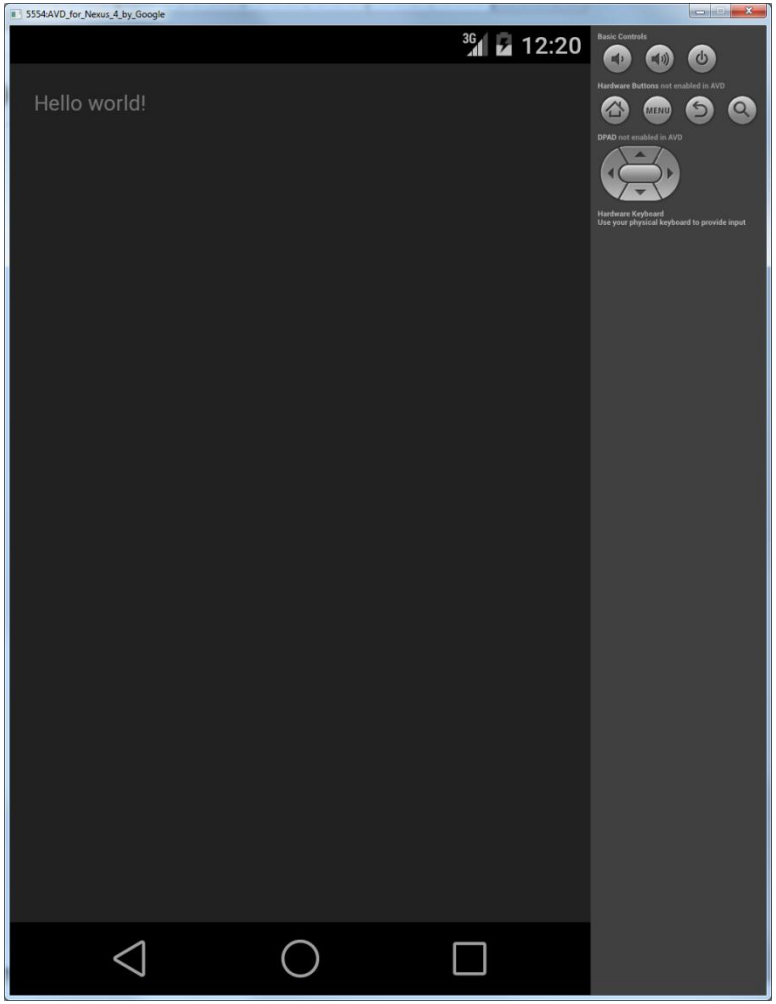


I assume you have already created an Android Virtual Device (emulator) using the AVD Manager. In this case, we'll launch the Nexus 4 emulator.

Auto-Generated Code Works!



The emulator may have its lock screen on. If that's the case, swipe up with your mouse to unlock the device and your app should be open. If you still don't see it, click the 'square' at the bottom and you should see all open apps.



The app works! This is one step further than we got in class on Thursday because of a configuration error on my Mac! ☹

OK, now let's build a custom view. There are two ways to approach the initial View class construction:

1. Manually
2. Via Android Studio's auto-generated code templates

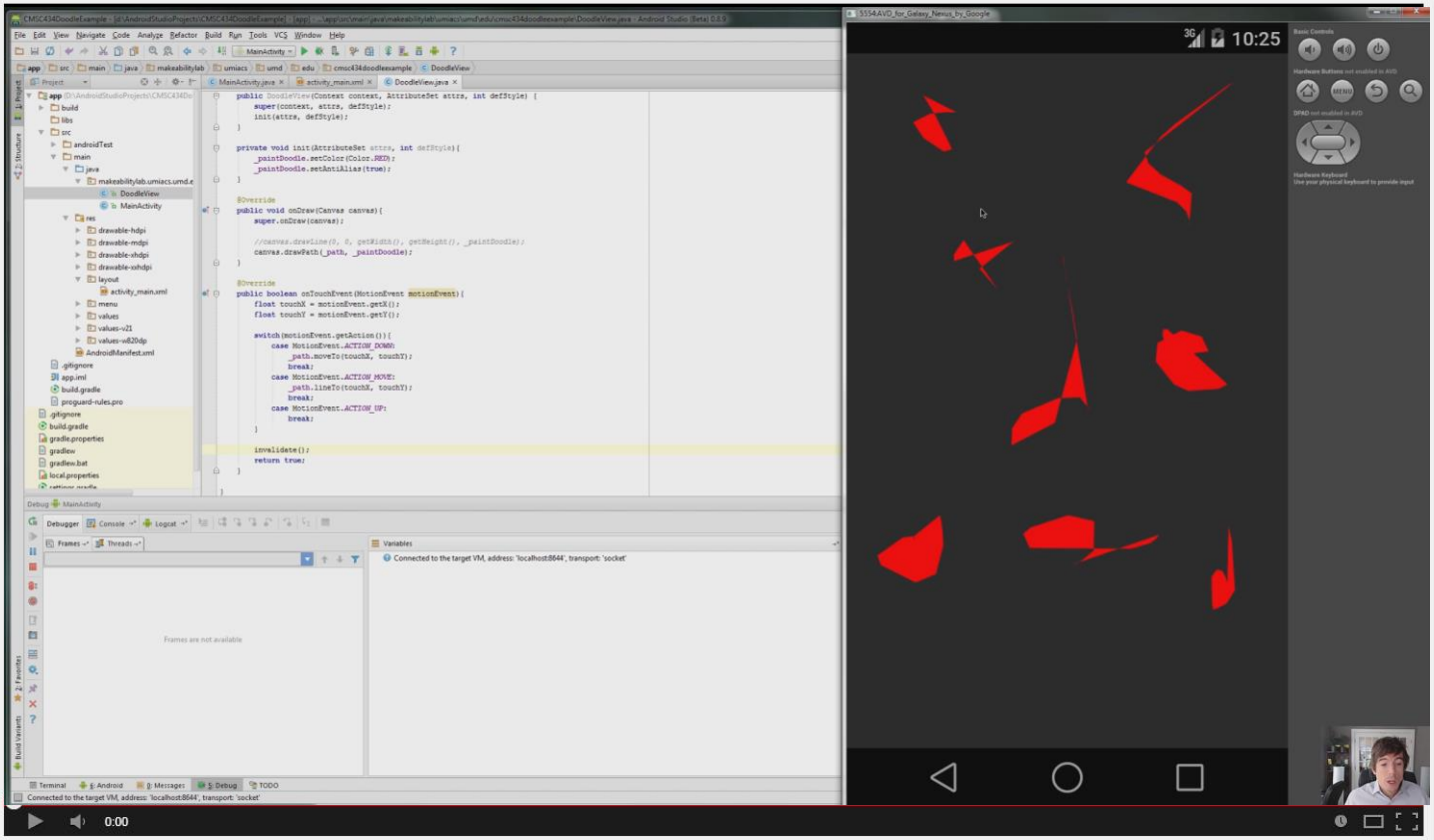
OK, now let's build a custom view. There are two ways to approach the initial View class construction:

1. Manually
2. Via Android Studio's auto-generated code templates

This is actually what we did in class but the auto-generated code is unnecessary and confusing. So, we'll start with #1 Manually.

OK, now let's build a custom view. There are two ways to approach the initial View class construction:

1. Manually
2. Via Android Studio's auto-generated code templates



[Click here to view: http://youtu.be/ktbYUrlN_Ws](http://youtu.be/ktbYUrlN_Ws)

Analytics | Video Manager

Quick Intro to Creating a Custom View in Android

Jon Froehlich

Channel settings

No views

Add to | Share | More

0 | 0

Share | Embed | Email

http://youtu.be/ktbYUrlN_Ws

Start at: 0:00

Uploaded on Oct 24, 2014

A quick introduction to creating a custom View in Android for my Introduction to Human-Computer Interaction class at the University of Maryland. Here, we create a View class that draws the user's touch locations and serves as a simple Doodle app. For more information, see the week 8 lectures here: http://cmsc434-f14.wikispaces.com/Lec…

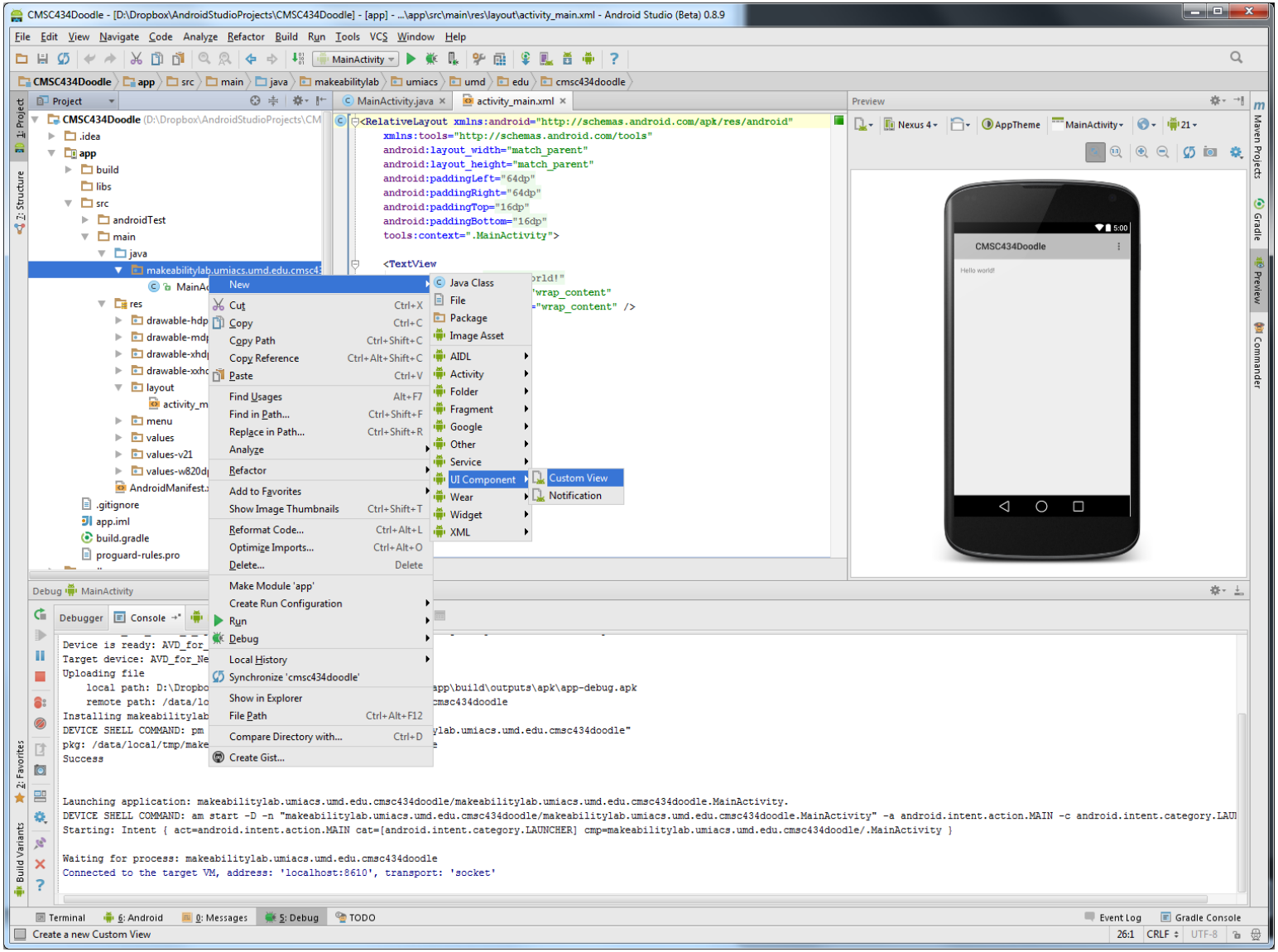
SHOW MORE

OK, now let's build a custom view. There are two ways to approach the initial View class construction:

1. Manually
2. Via Android Studio's auto-generated code templates

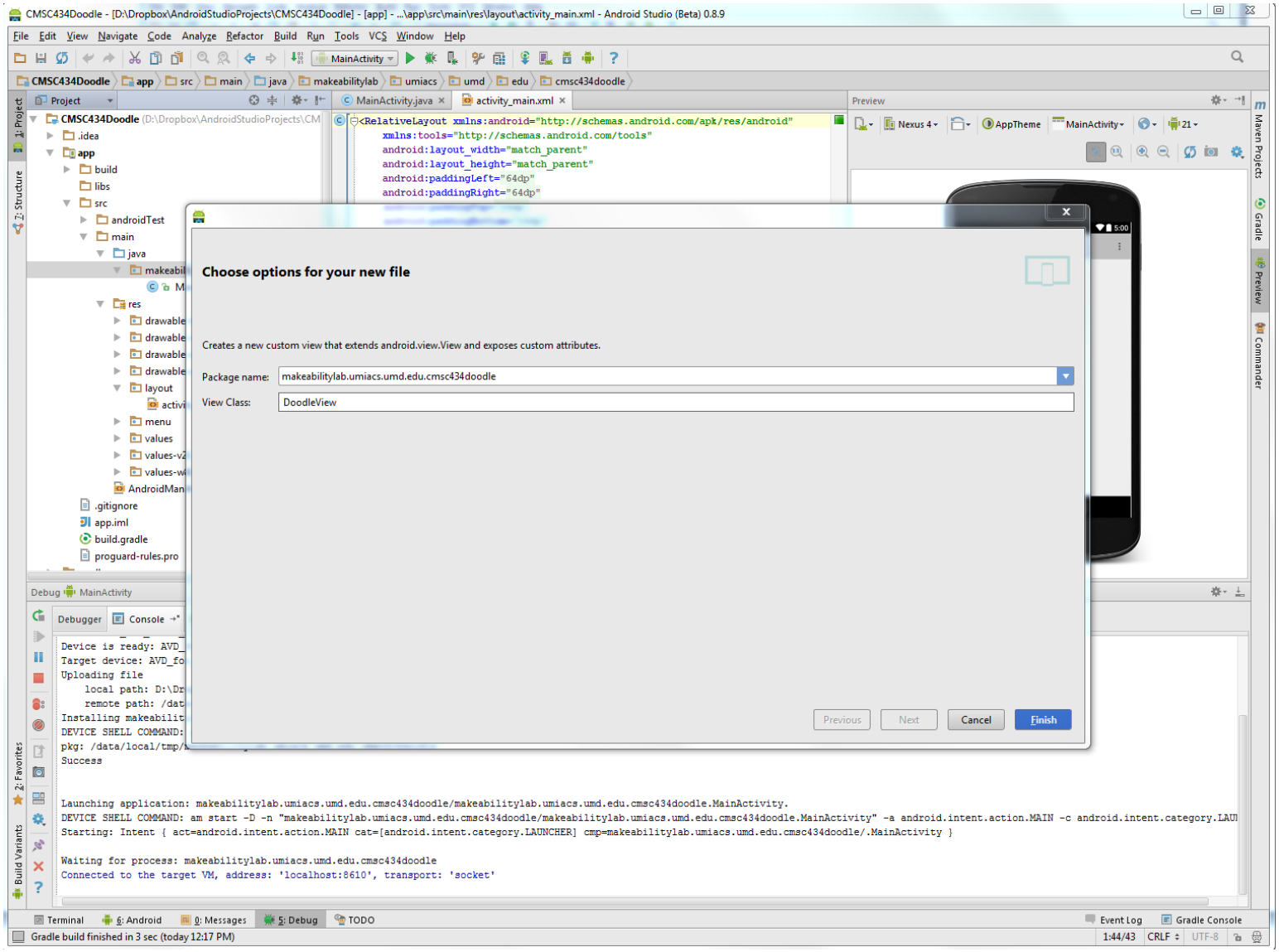
The advantage of this method is it shows you how to setup XML-based attributes/styles. This is more advanced then we need to get in class but is necessary to understand if you want to build a reusable View component. So, for those that are curious, read ahead!

New->UI Component->Custom View

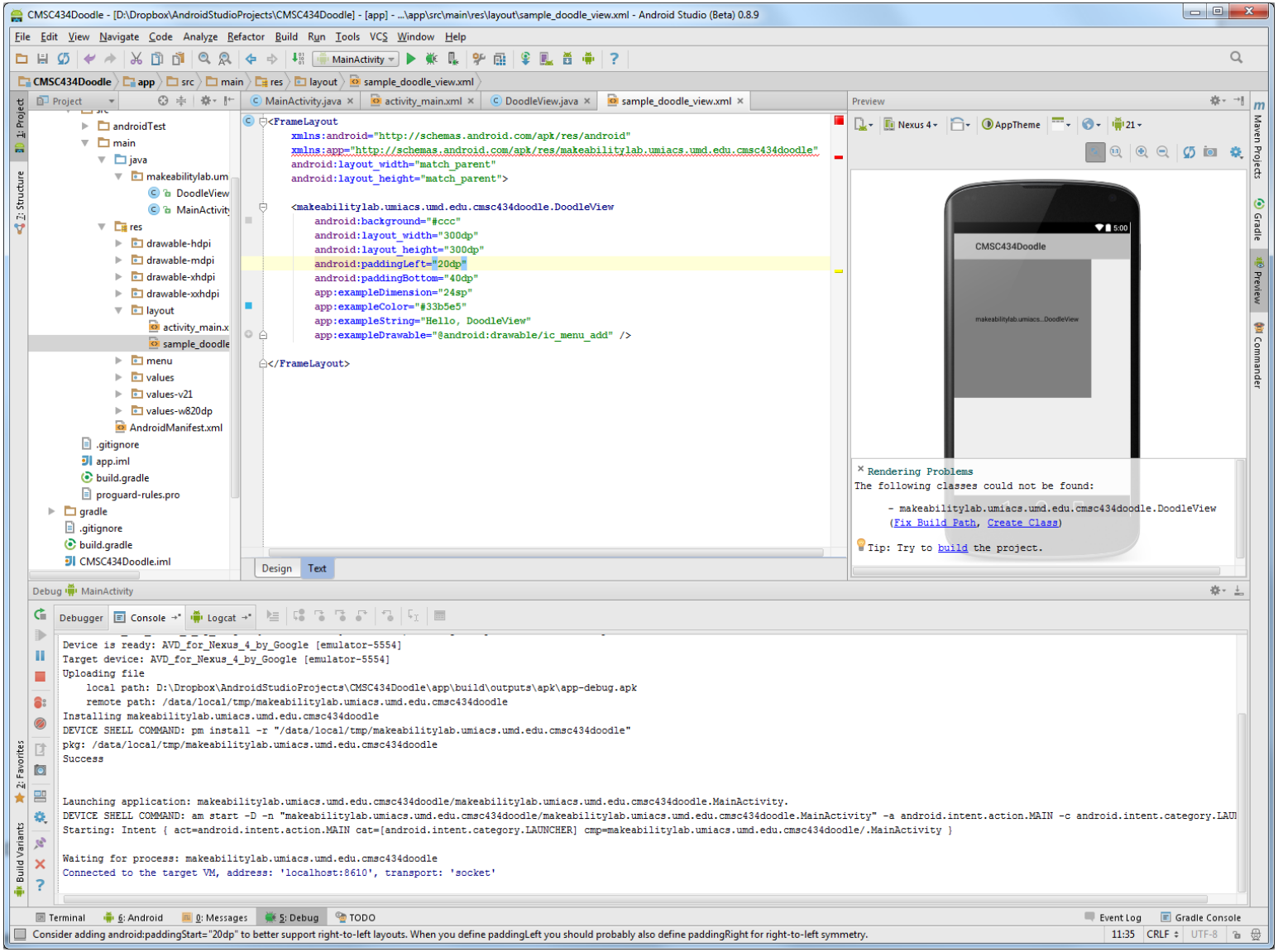


Here, we are going to use Android Studio's built in Custom View template to create the initial skeleton of our custom view. This will auto-generate some Java code, some XML, and add the view to the manifest.

NAME THE VIEW: DOODLEVIEW

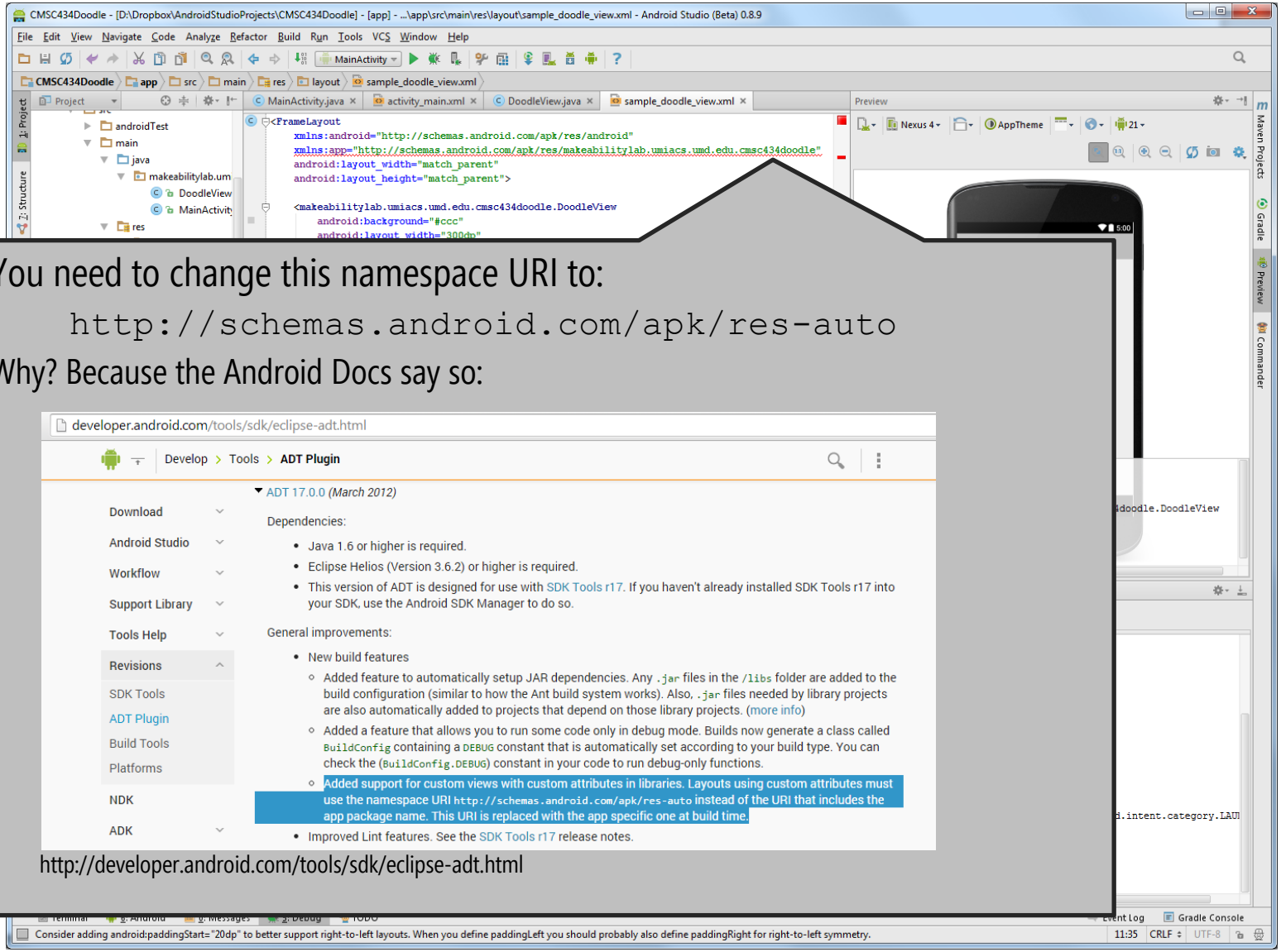


Uh, Oh! A Problem Already!



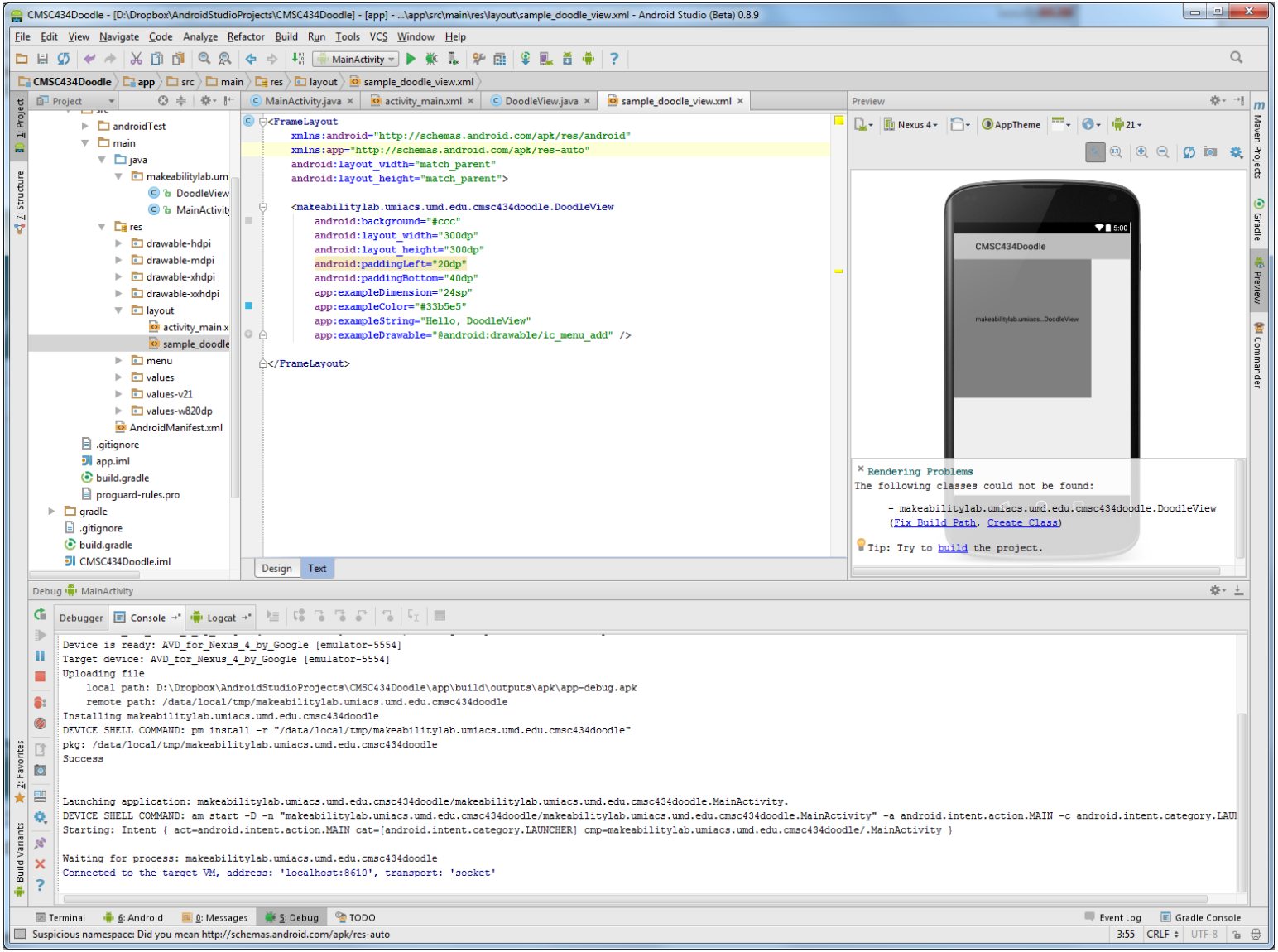
Uh oh, the auto-generated XML file contains an error! What's going on here?

Uh, Oh! A Problem Already!



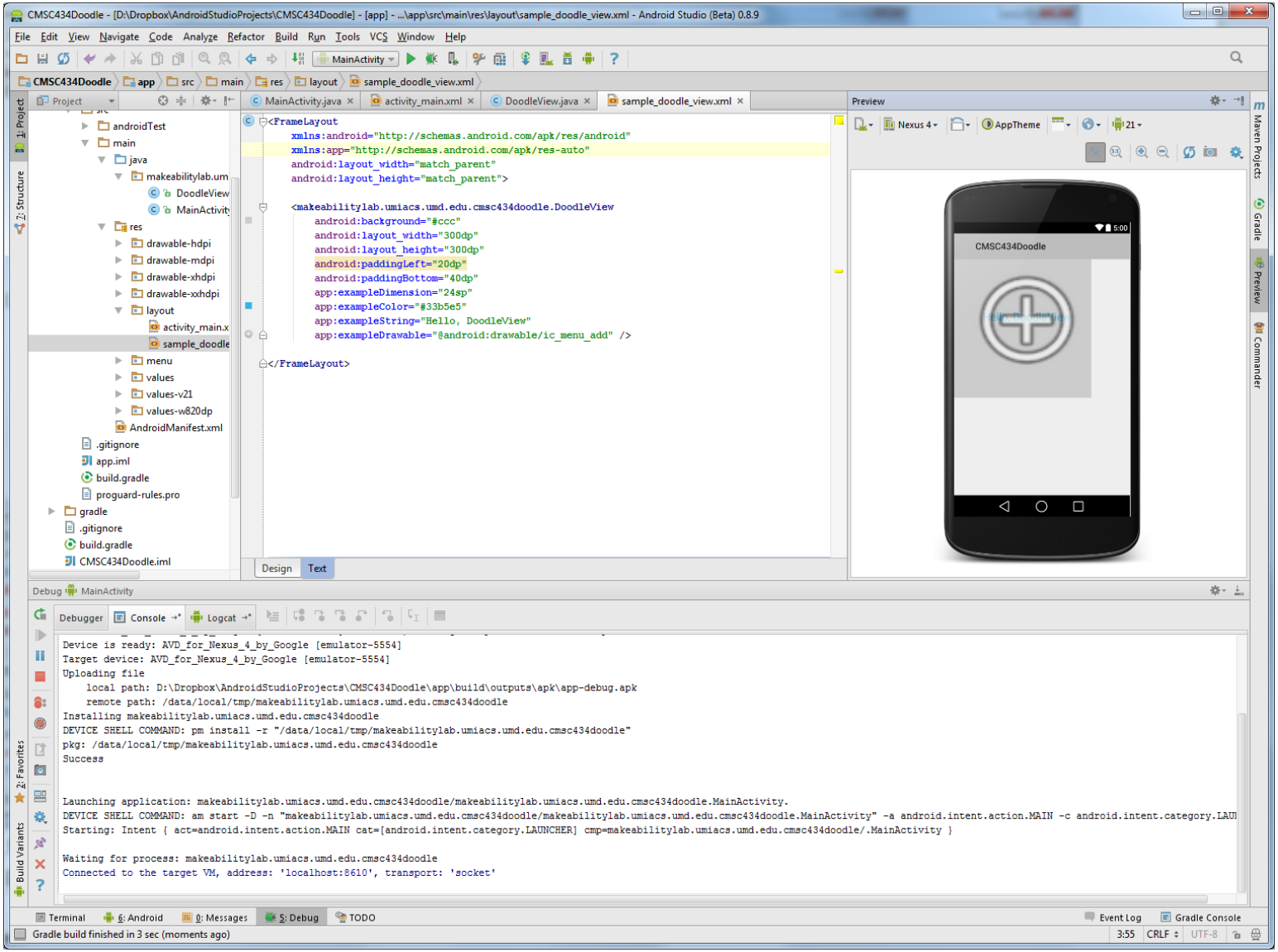
Uh oh, the auto-generated XML file contains an error! What's going on here?

Build the Project to Fix the Preview



OK, fixed that problem. Hmm, the "Preview" in the upper-right corner has a warning about "Rendering Problems." Let's try to build the project as the dialog suggests and see if it goes away.

Preview Pane Works!



Great, now we actually get to see what the DoodleView looks like now... the auto-generated code draws a dummy bitmap image, which is stretched across the widget and a dummy text string labeled 'Hello, DoodlView.' This code is auto-generated to help you, as a developer, understand how to build custom views but we don't actually need it. We'll go through it briefly before deleting it.

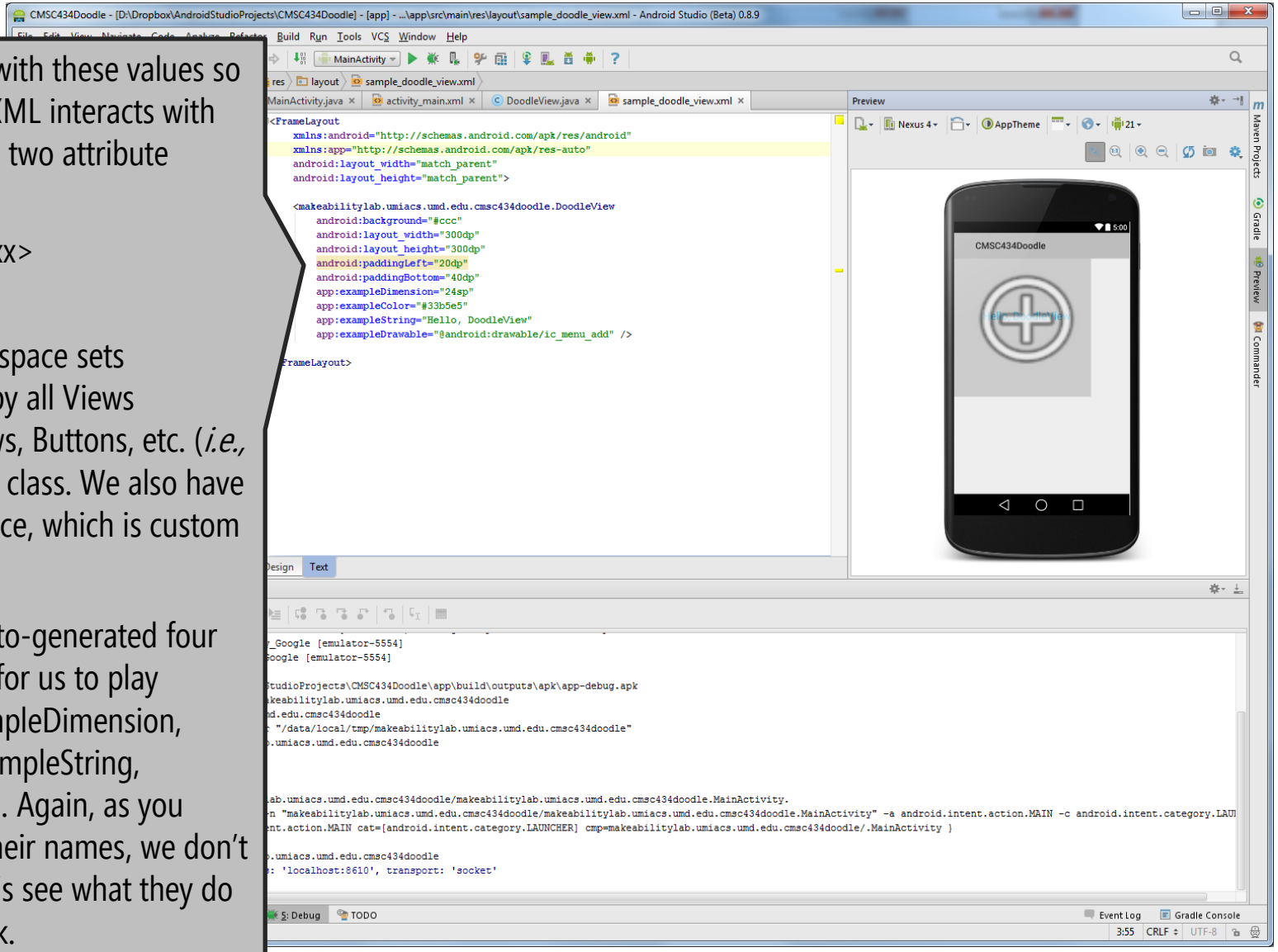
View Attributes: Overview

Let's play around with these values so you can see how XML interacts with our app. There are two attribute namespaces here:

1. android:<xxxxx>
2. app:<xxxxx>

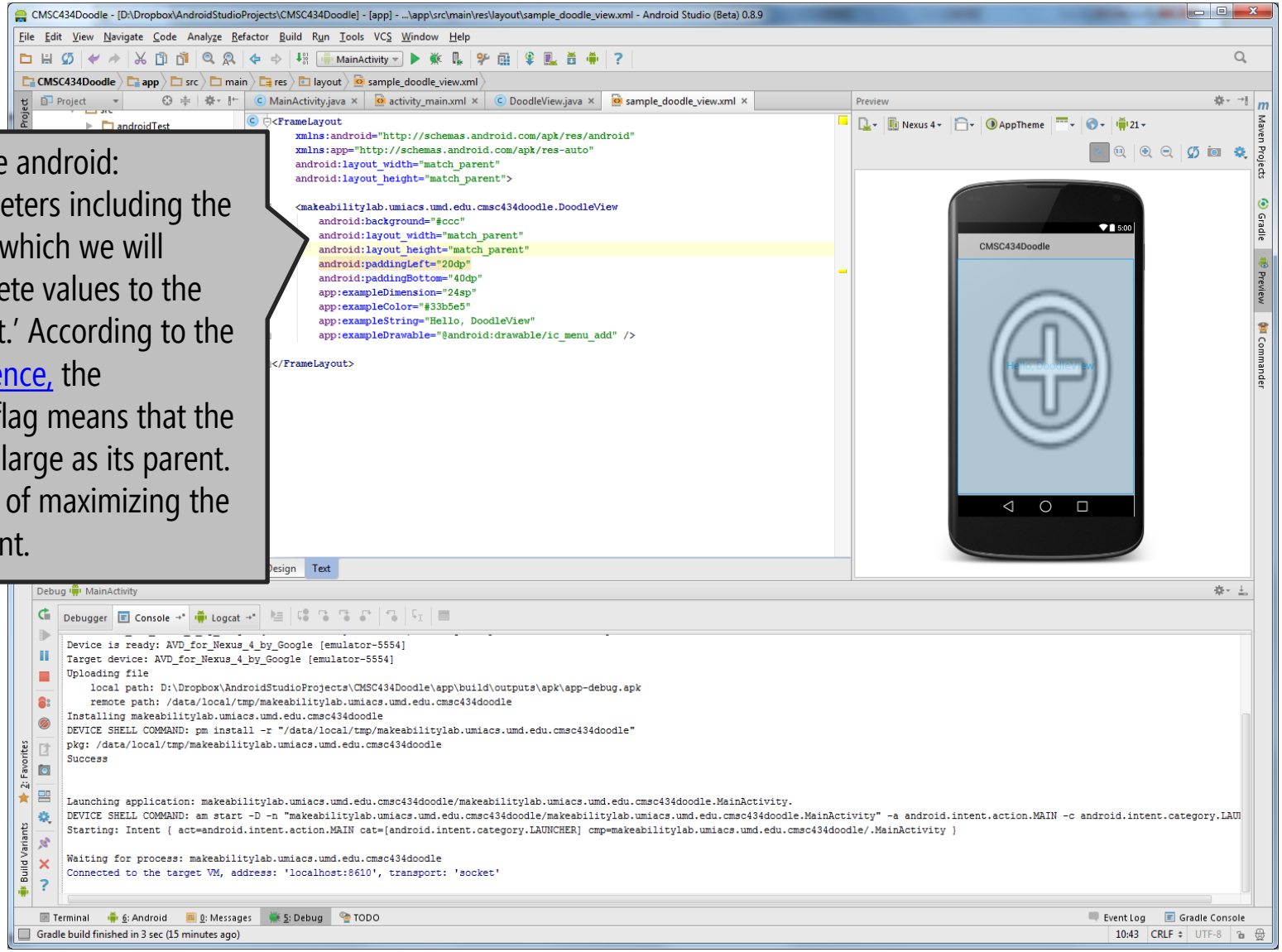
The android namespace sets attributes shared by all Views including TextViews, Buttons, etc. (*i.e.,* in the parent View class. We also have the 'app' namespace, which is custom to our app.

Android Studio auto-generated four custom attributes for us to play around with (exampleDimension, exampleColor, exampleString, exampleDrawable). Again, as you could infer from their names, we don't need these but let's see what they do and how they work.



View Attributes: Match_Parent

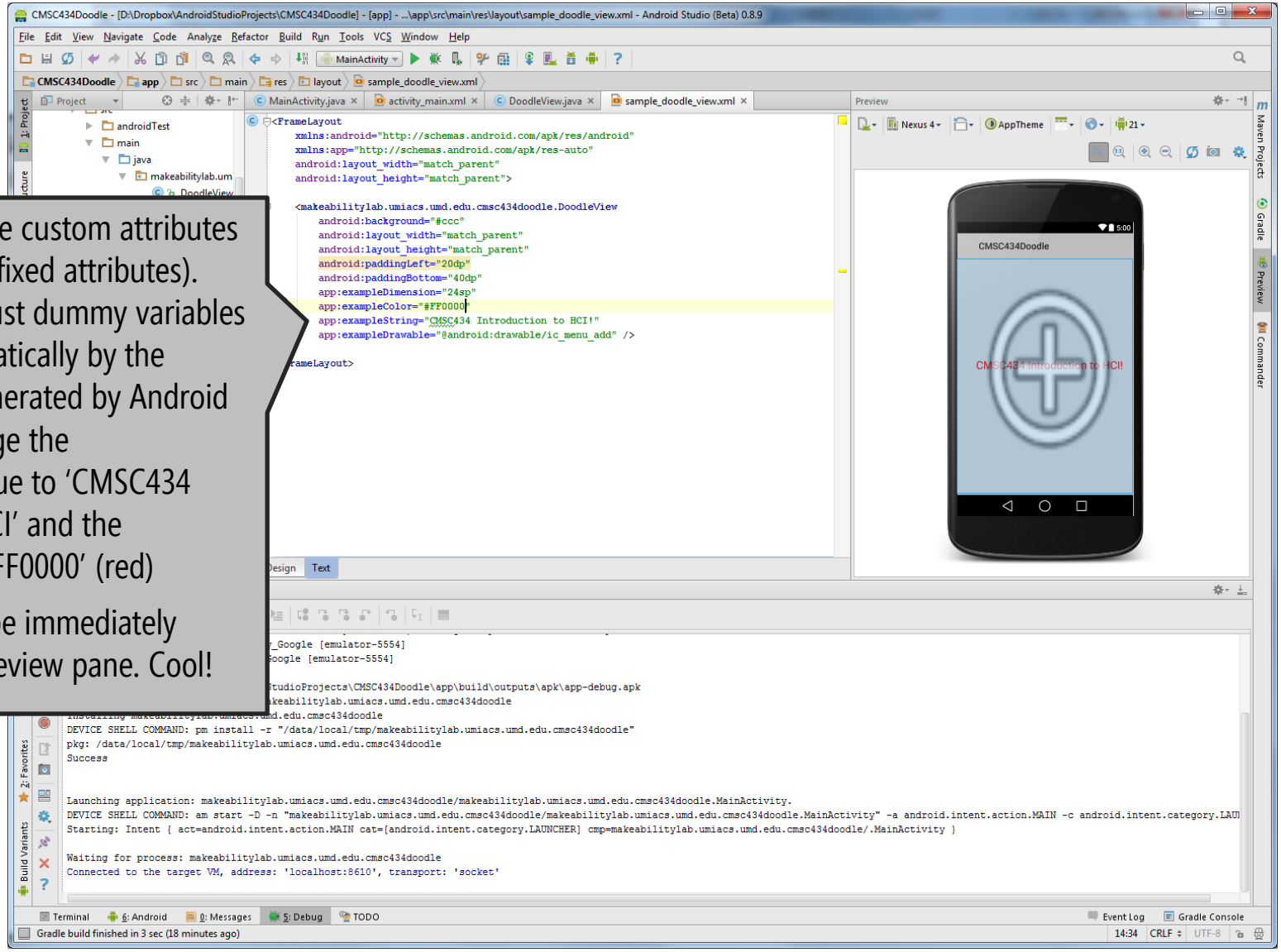
First, let's set some android: namespace parameters including the width and height, which we will change from discrete values to the flag 'match_parent.' According to the Android API reference, the MATCH_PARENT flag means that the view should be as large as its parent. This has the effect of maximizing the view to fill its parent.



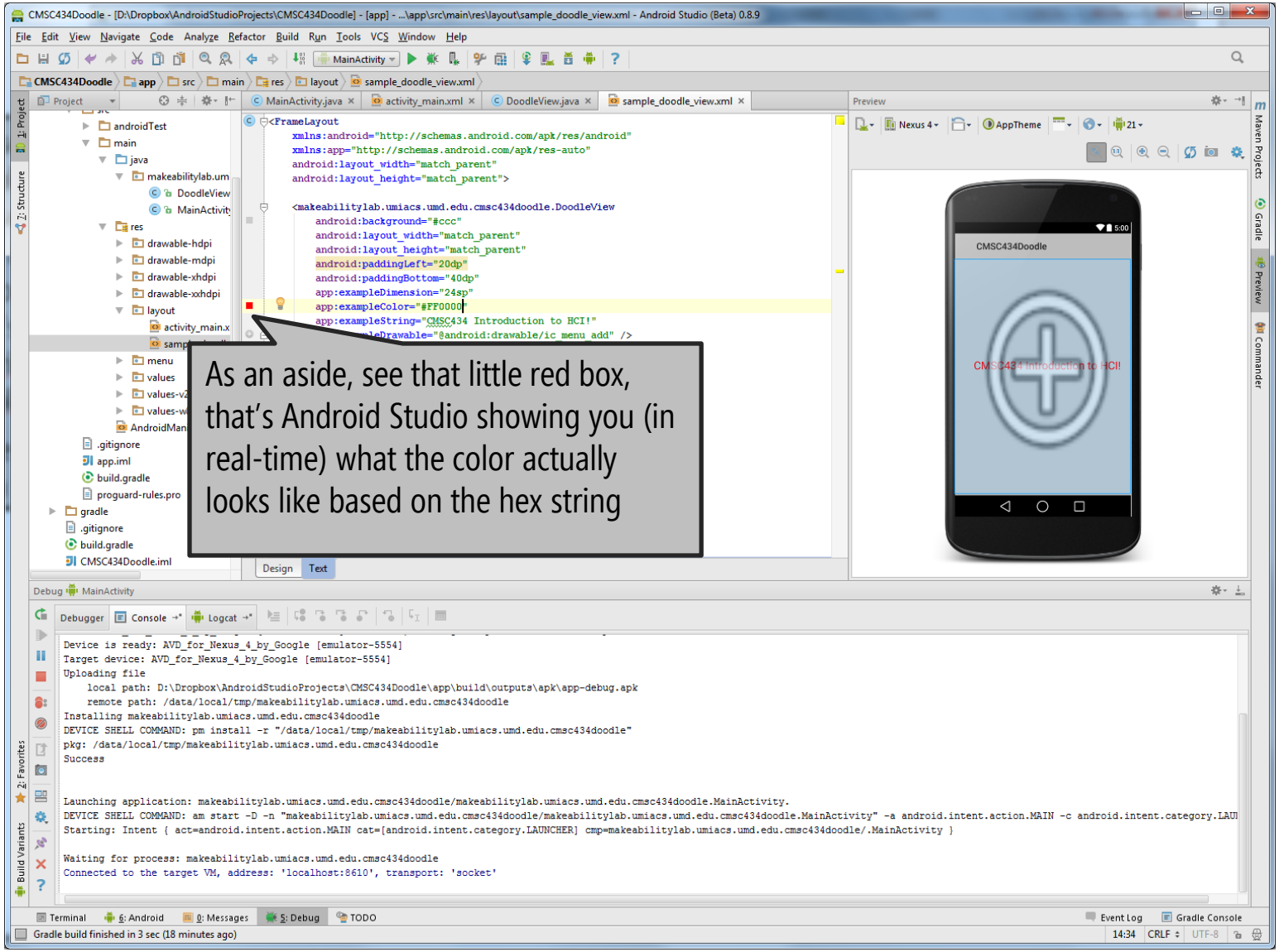
View Attributes: App Specific

Now, let's set some custom attributes (the example* prefixed attributes). Again, these are just dummy variables hooked up automatically by the template code generated by Android Studio. Let's change the exampleString value to 'CMSC434 Introduction to HCI' and the exampleColor to 'FF0000' (red)

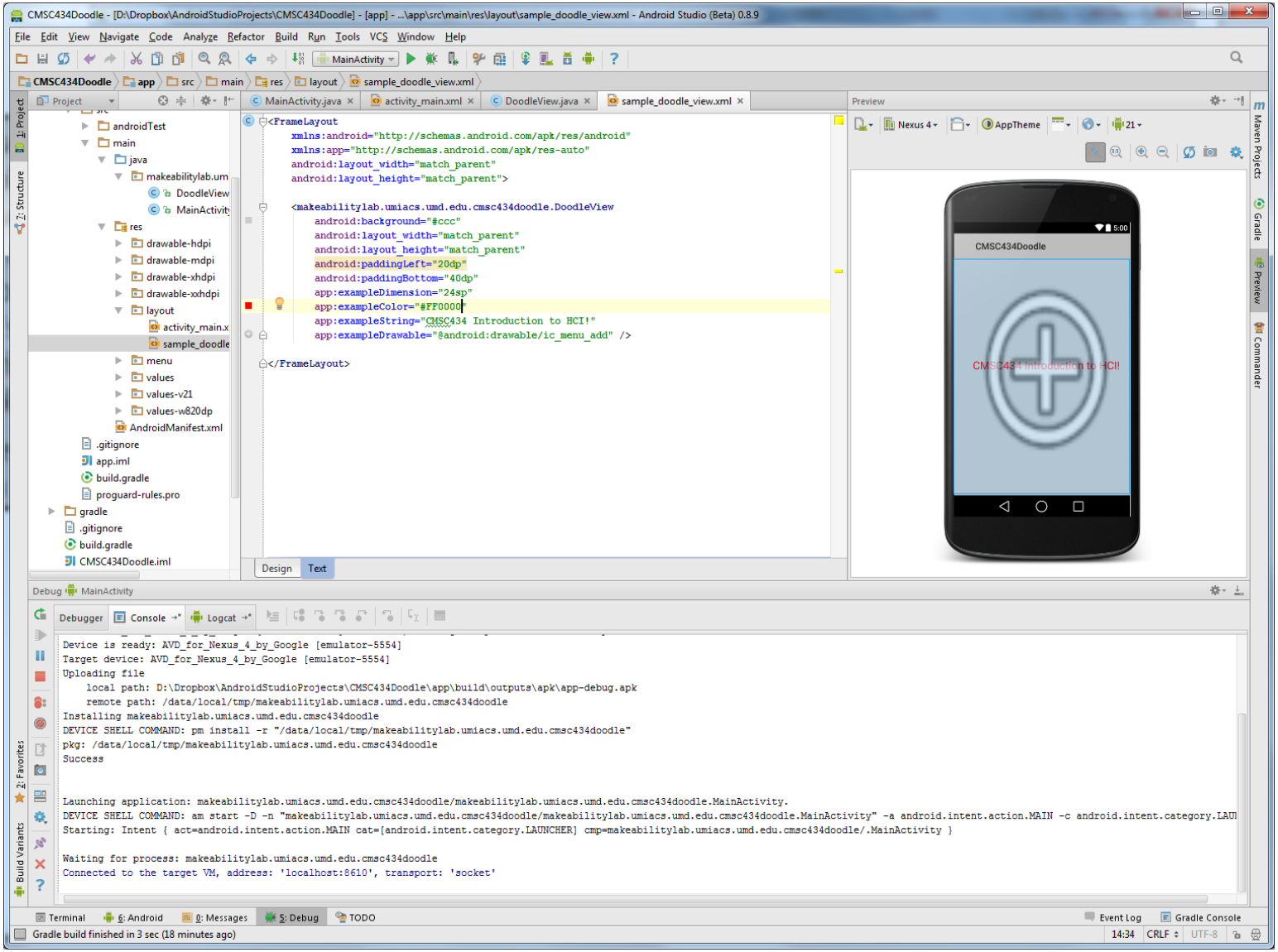
The changes will be immediately reflected in the Preview pane. Cool!



View Attributes: App Specific



View Attributes: How Does This Work?



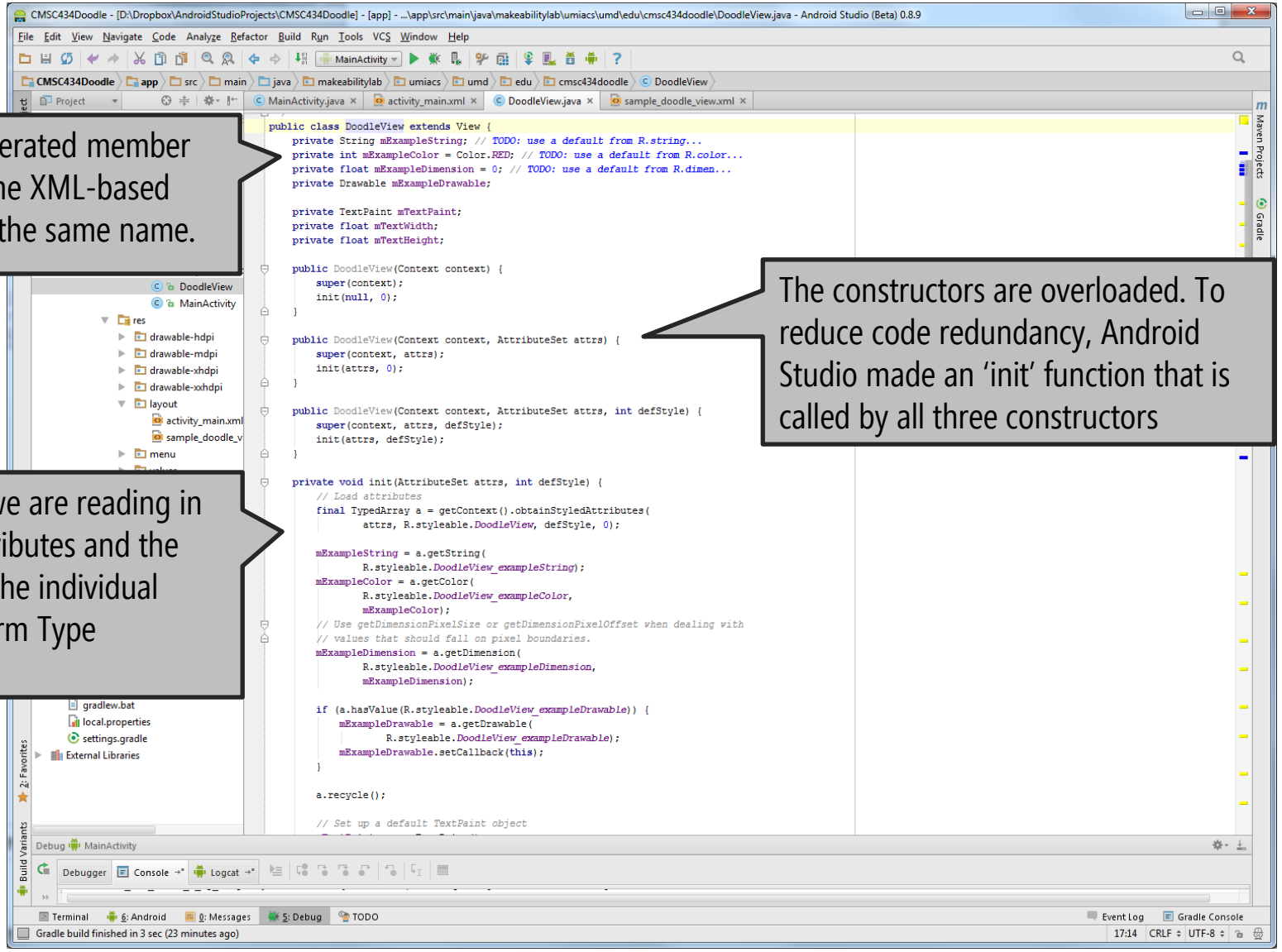
OK, but what's really happening when we set these attributes? Why is it changing our custom view? For this, we have to explore the auto-generated Java code that Android Studio made for us.

View Attributes: How Does This Work?

These four auto-generated member variables will store the XML-based attributes that carry the same name.

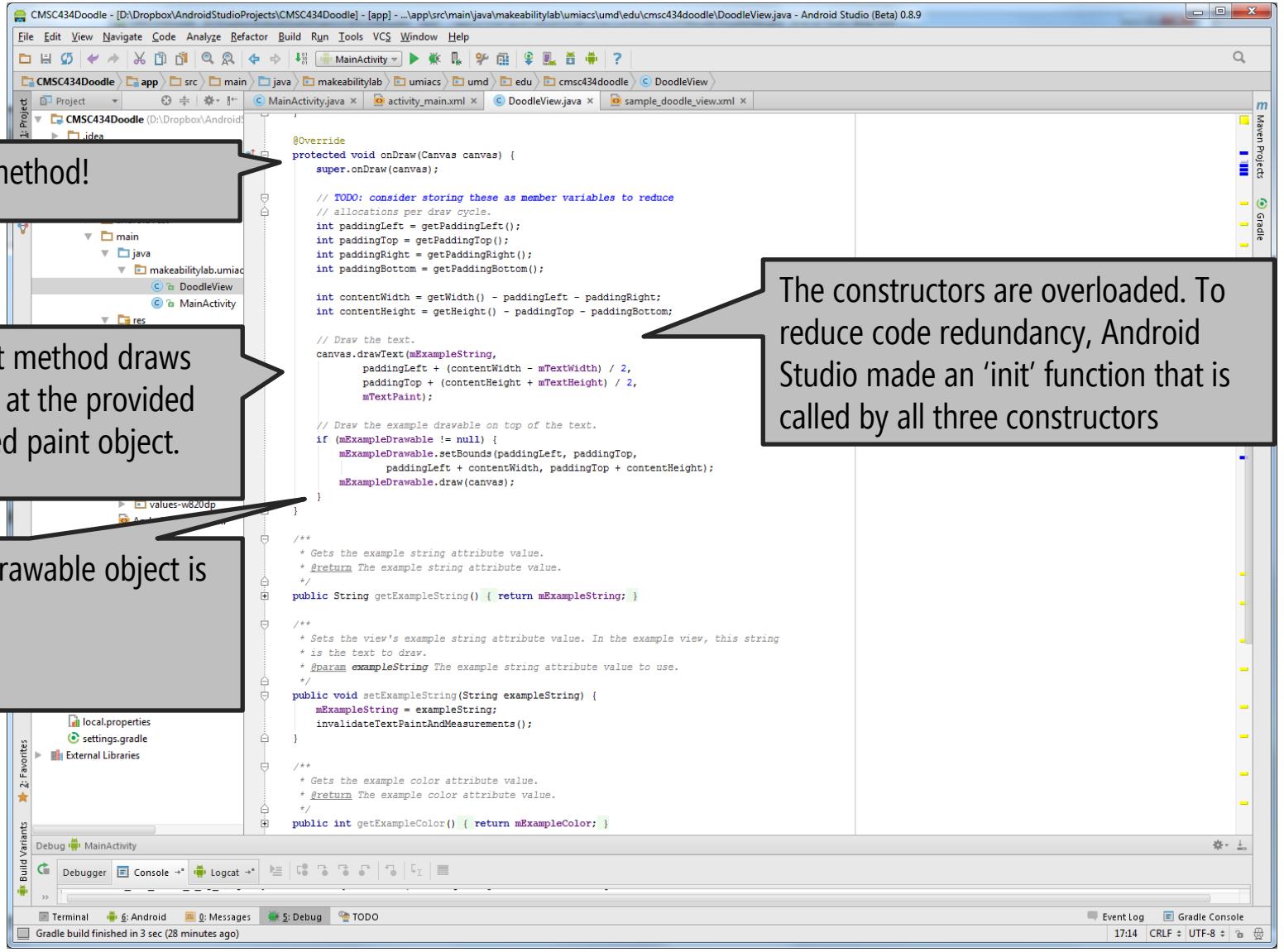
The constructors are overloaded. To reduce code redundancy, Android Studio made an 'init' function that is called by all three constructors

Aha, here is where we are reading in the custom XML attributes and the next few lines grab the individual attributes and perform Type conversion.



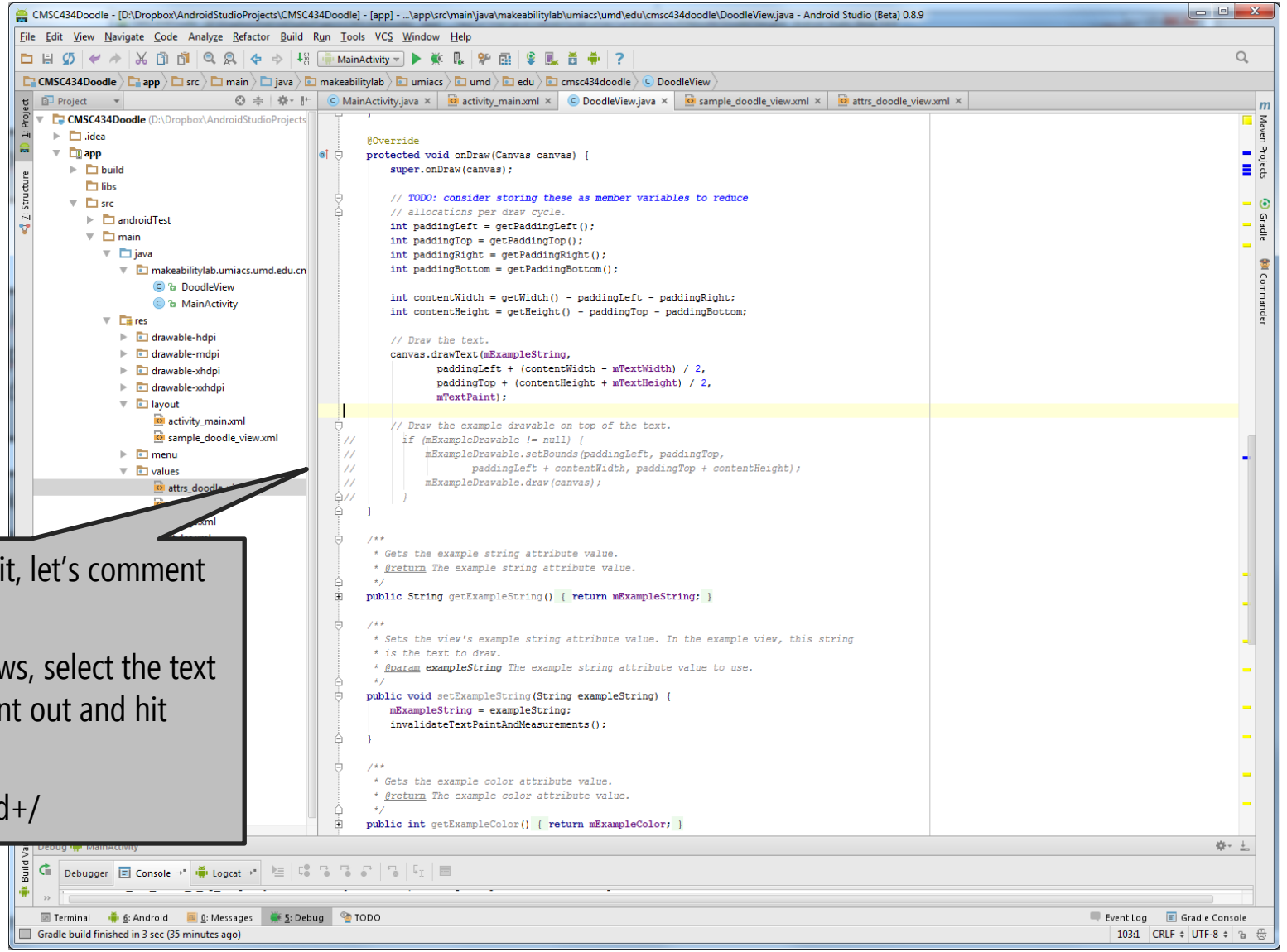
This shows you how to read in custom attributes and set them to member variables… but the question still remains: how are they used? For this, we have to look at the onDraw method. This is where all (most) of the fun happens in a custom view.

View Attributes: How Does This Work?



This shows you how to read in custom attributes and set them to member variables… but the question still remains: how are they used? For this, we have to look at the onDraw method. This is where all (most) of the fun happens in a custom view.

View Attributes: How Does This Work?

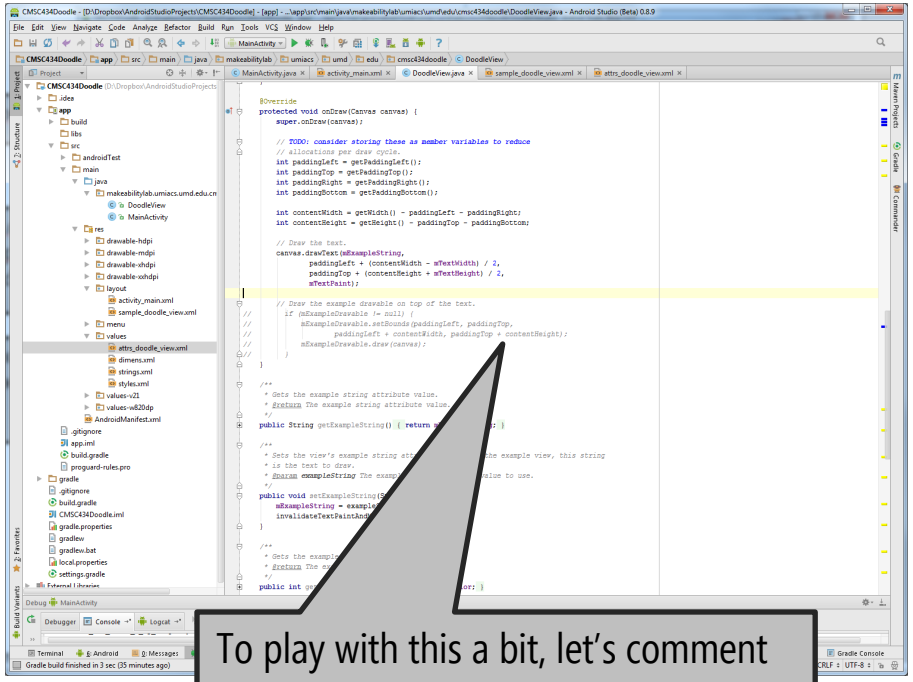


To play with this a bit, let's comment this out.

Quick tip: on Windows, select the text you want to comment out and hit ctrl+/

On a Mac, press cmd+/

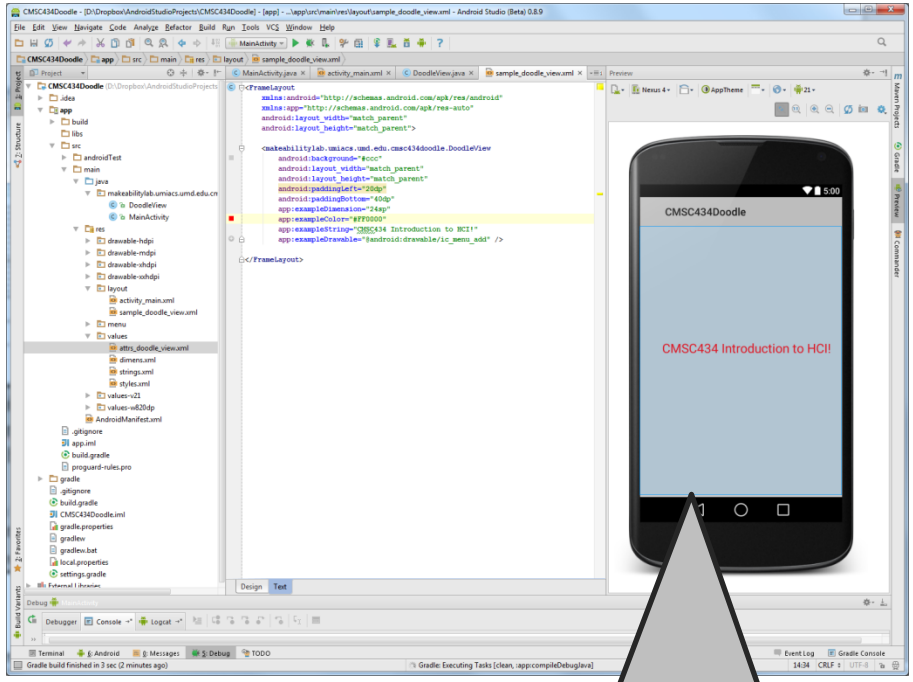
View Attributes: How Does This Work?



To play with this a bit, let's comment the mExampleDrawable code out and see what happens in our preview pane

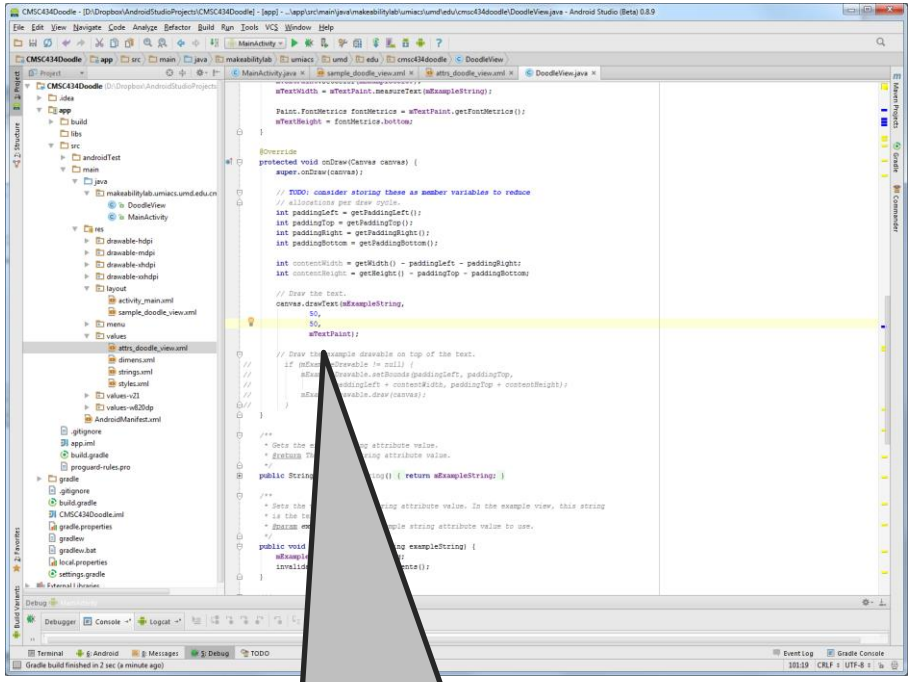
Quick tip: on Windows, select the text you want to comment out and hit ctrl+/

On a Mac, press cmd+/

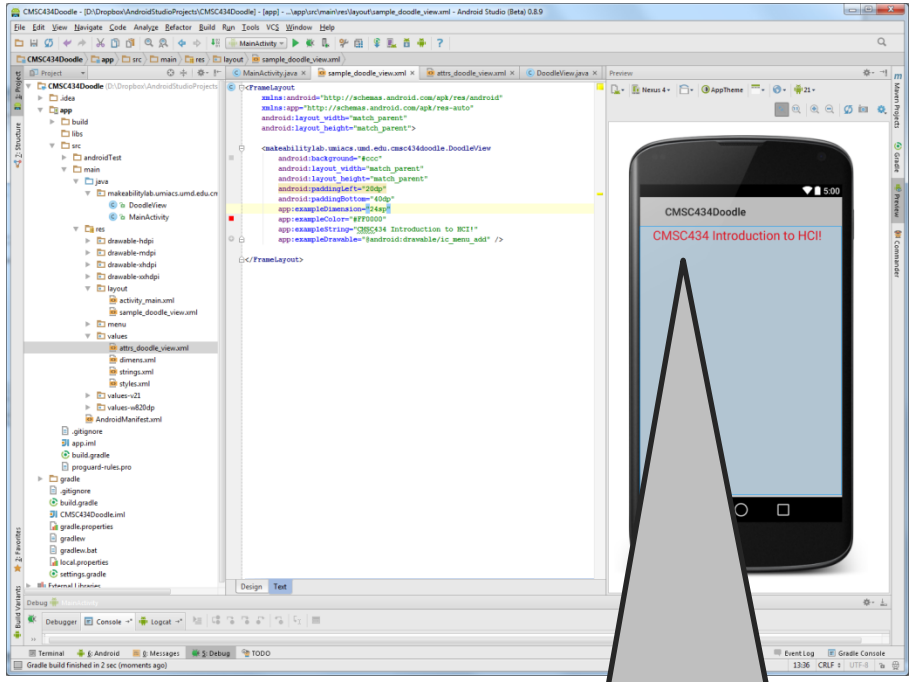


The stretched icon is gone! So, cool—even the programmatic code I change in Java is reflected in the preview pane. Let's try another change.

View Attributes: How Does This Work?



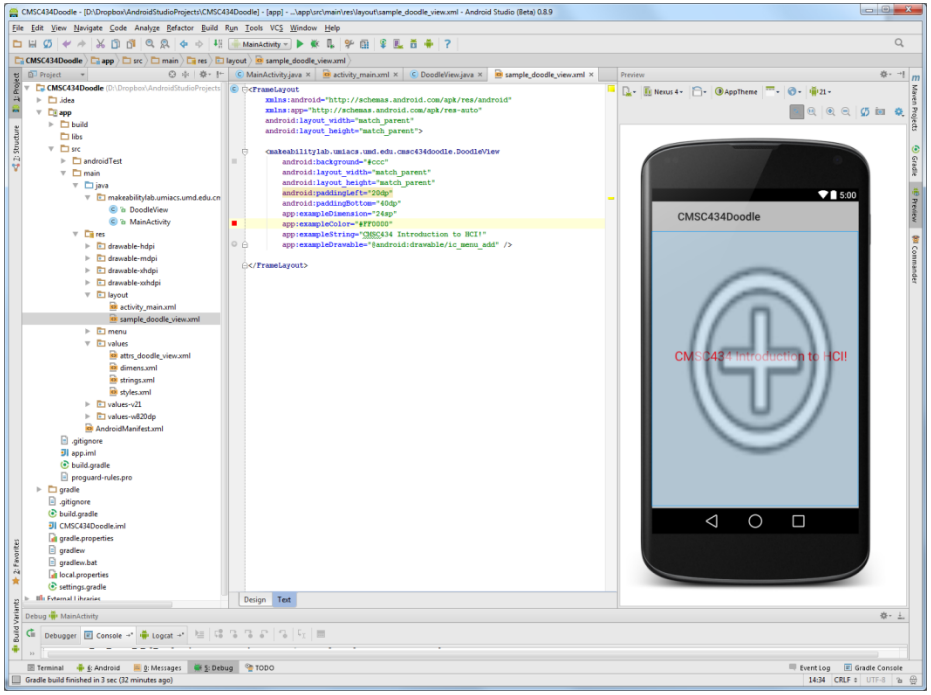
Let's change the location of where the text is drawn to x=50, y=50



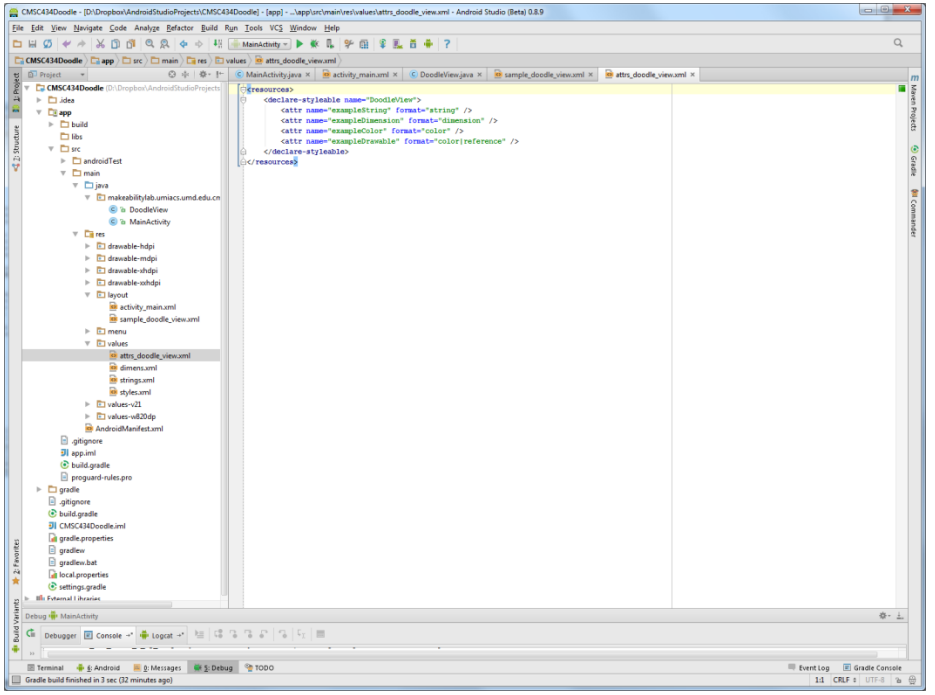
And here we are! A new location of the text.

Where to Define Custom Attributes in XML?

The final relevant point to cover about attributes is where to define our custom attributes for our View. They are stored in values->sample_doodle_view.xml. See below.



The XML file for our custom view (sample_doodle_view.xml)

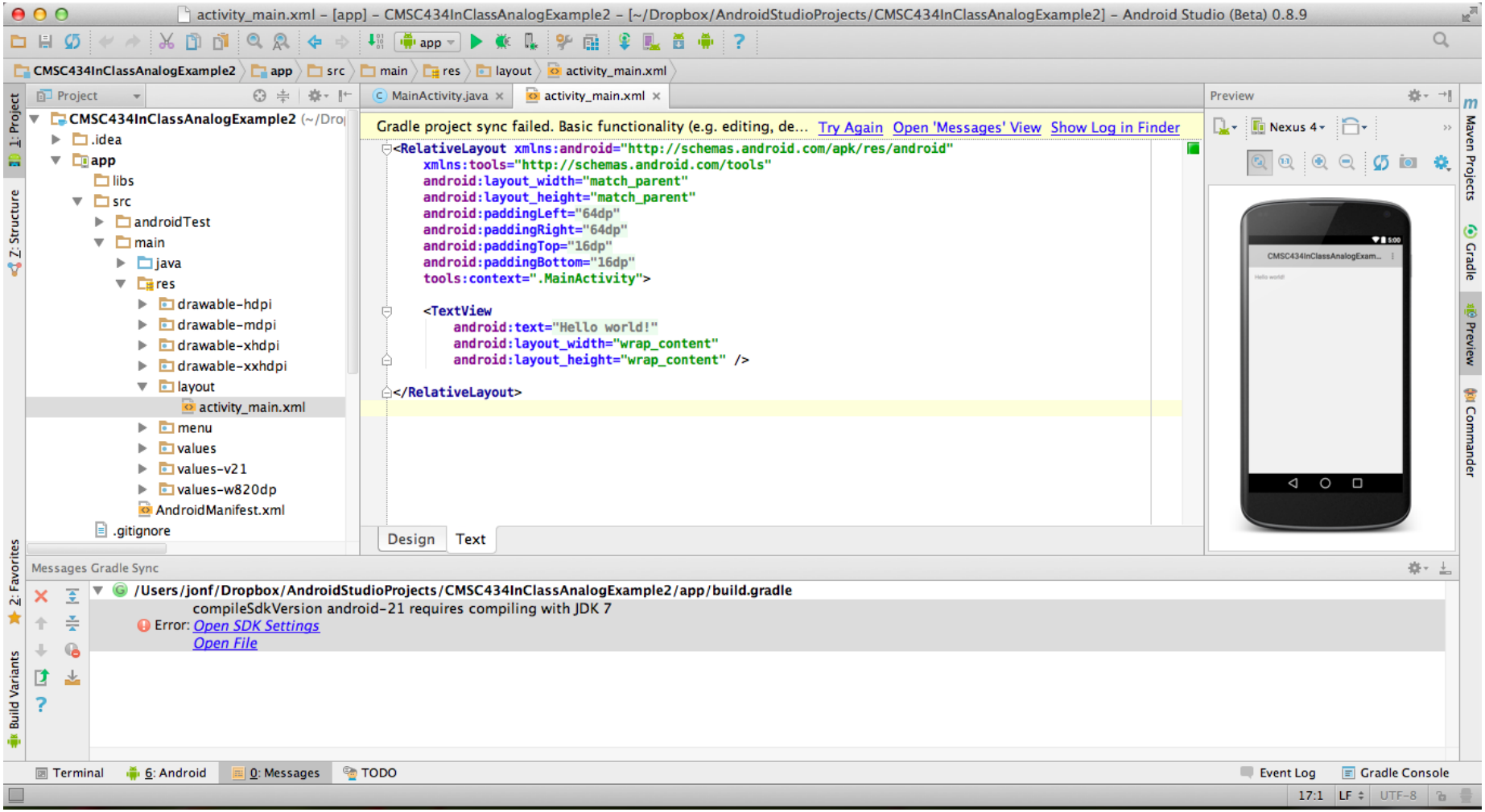


The auto-generated attributes file for our custom view with the four example attributes and their type. If you wanted to make more custom attributes for DoodleView, you'd put them here!

Fixing Gradle Project Sync Failed Problem

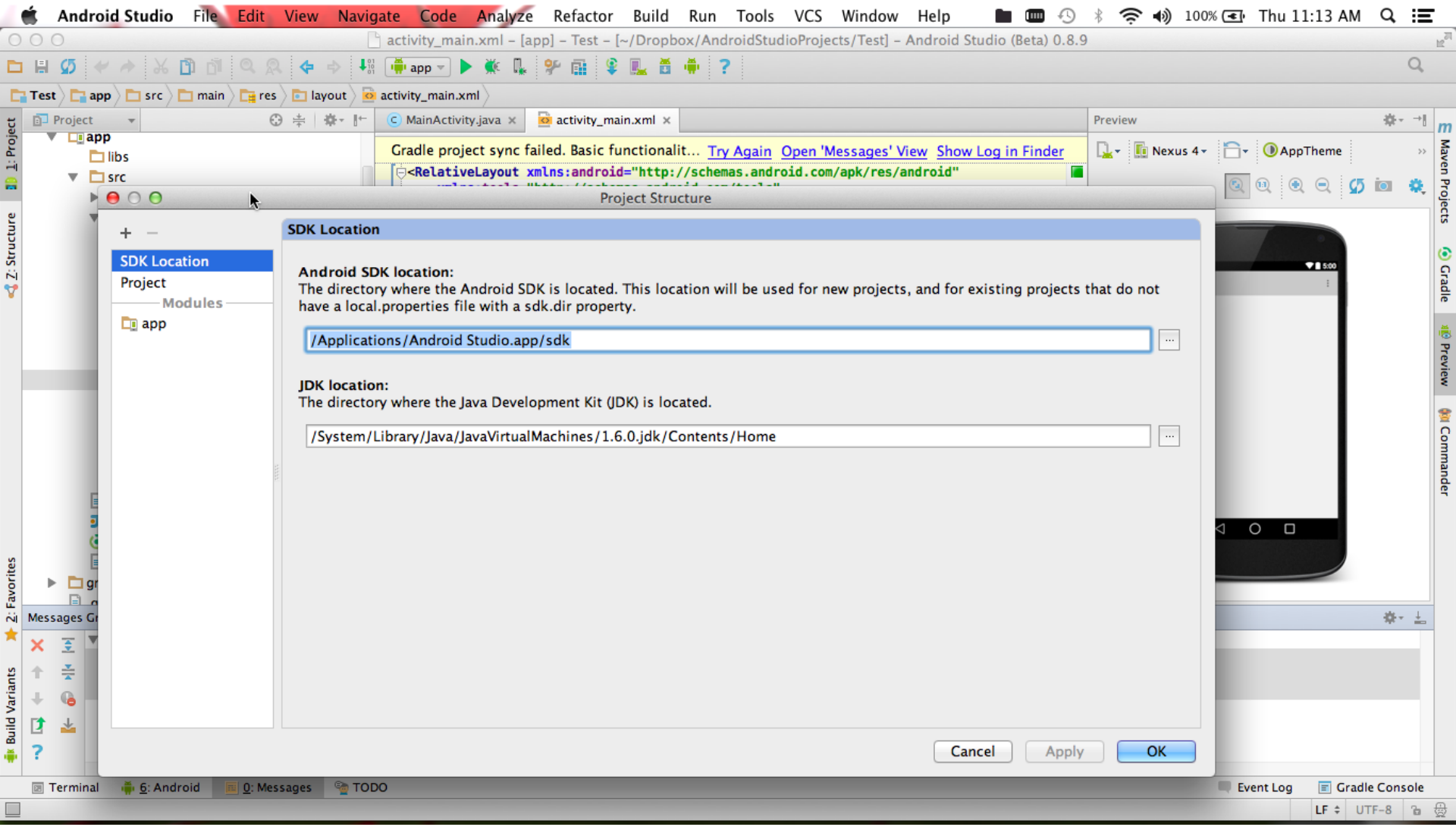
The following slides should help you fix the config problem we ran into during class on my Mac. A few others had this issue as well (*e.g.,* Joel).

Gradle Project Sync Failed Error



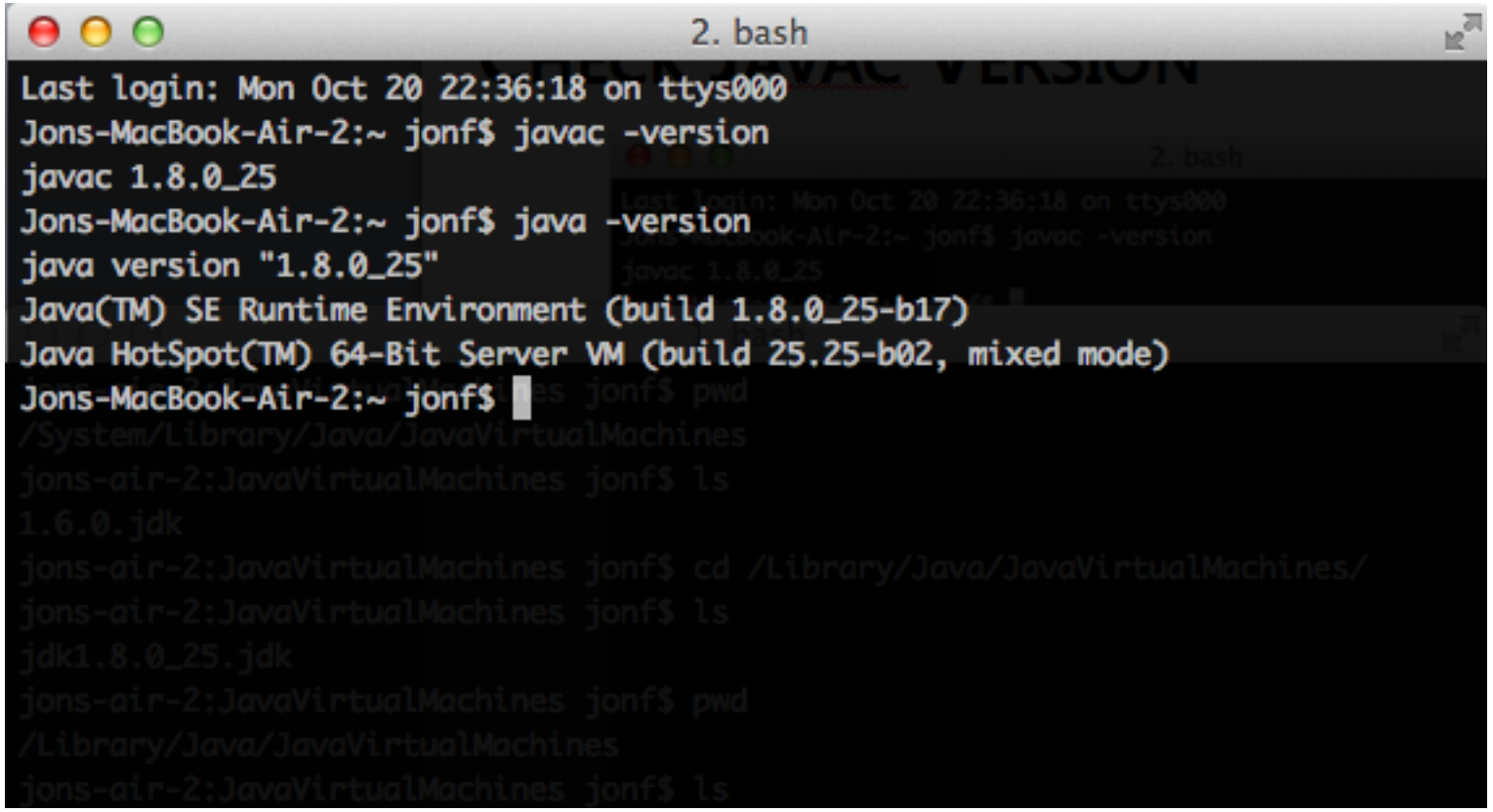
The error shows up at the top of the screen (in the yellow bar). If you click "Open 'Messages' View" then more details show in the bottom Messages pane including the root of our issue: we need to compile Android API 21 with JDK 7 or later. So, let's investigate. Click on "Open SDK Settings"

The SDK Location Dialog



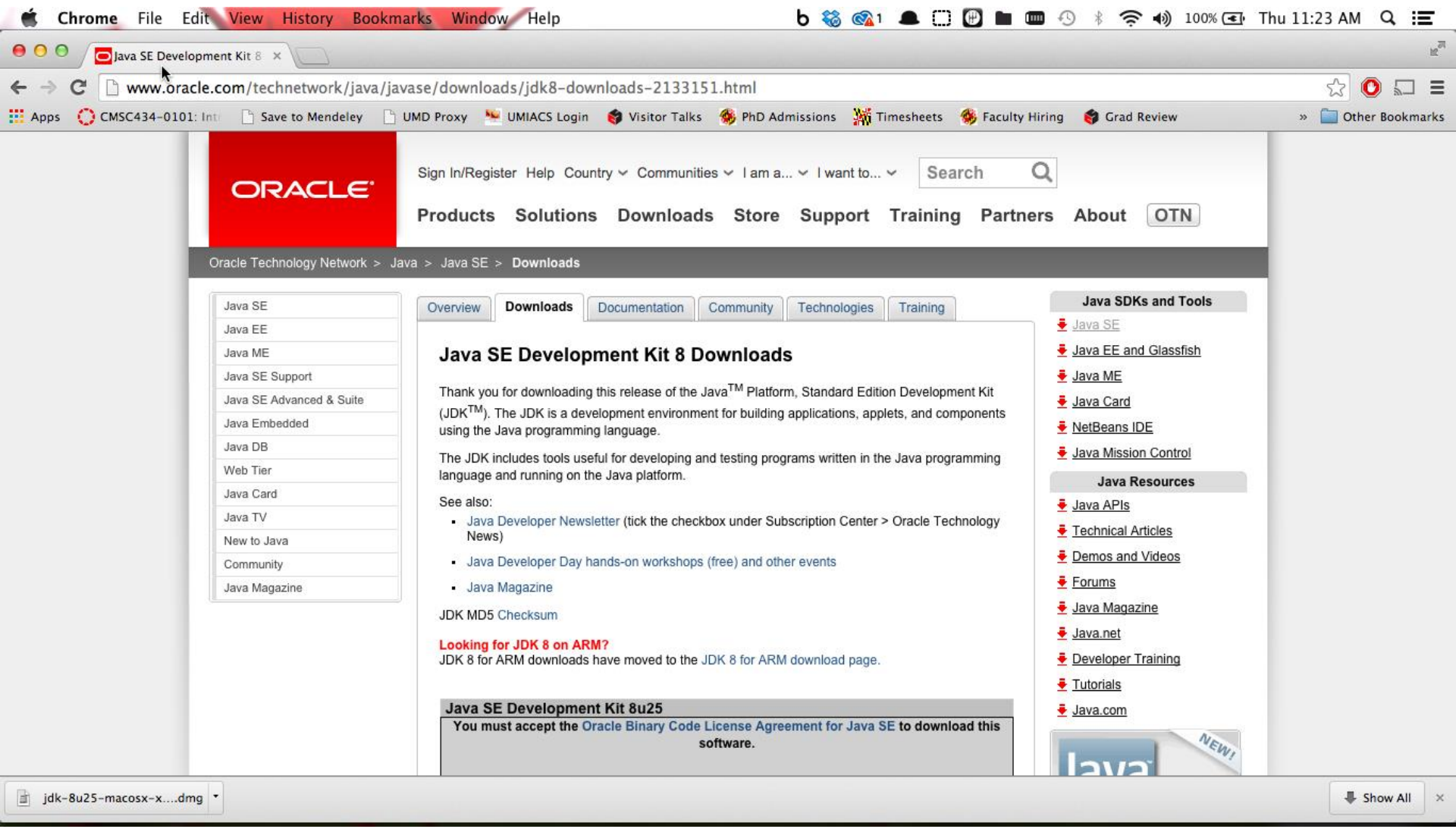
The problem is that the JDK location is pointing to 1.6 rather than 1.7 (or later), we need to change this!

Check Javac Version



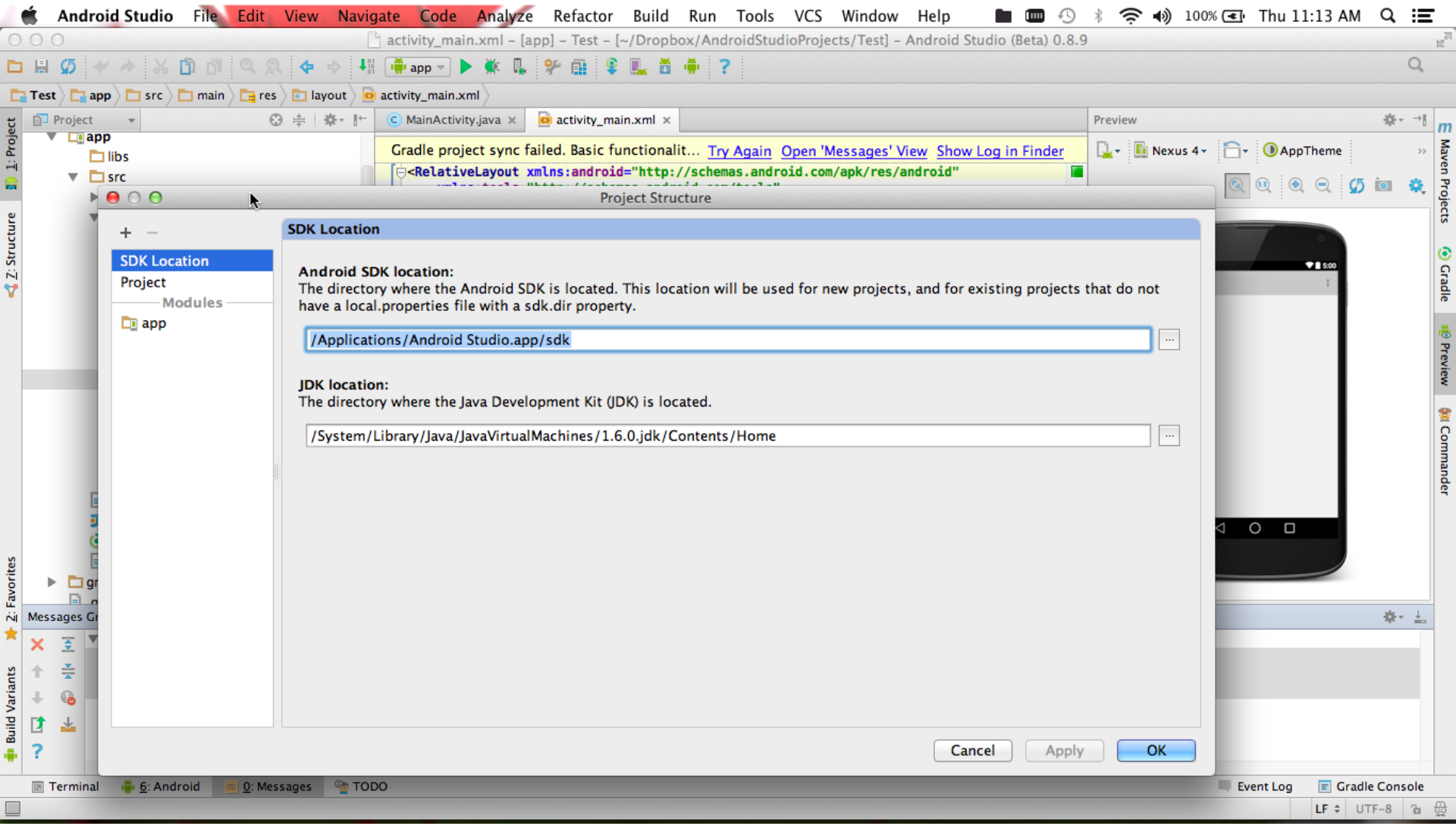
Open terminal and check to see which 'javac' version you are using. See, you can tell that I already have JDK8 installed; so, the problem simply was that the JDK location path was wrong in my project settings (it defaults to the wrong path).

DOWNLOAD JDK 7 OR LATER

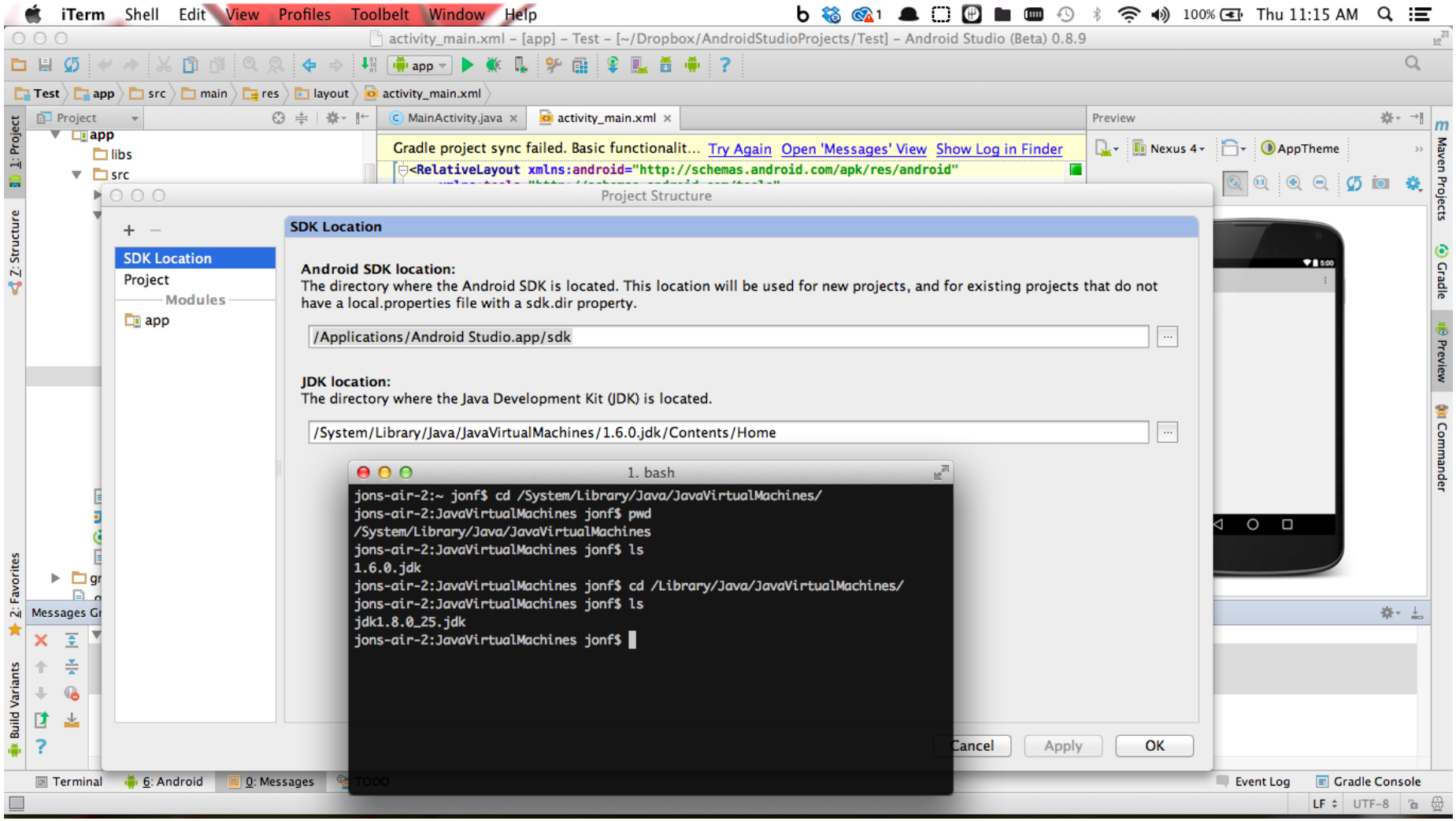


If you don't have JDK7 installed or later, go to the Oracle website to download the newest JDK.

Determine New JDK Location

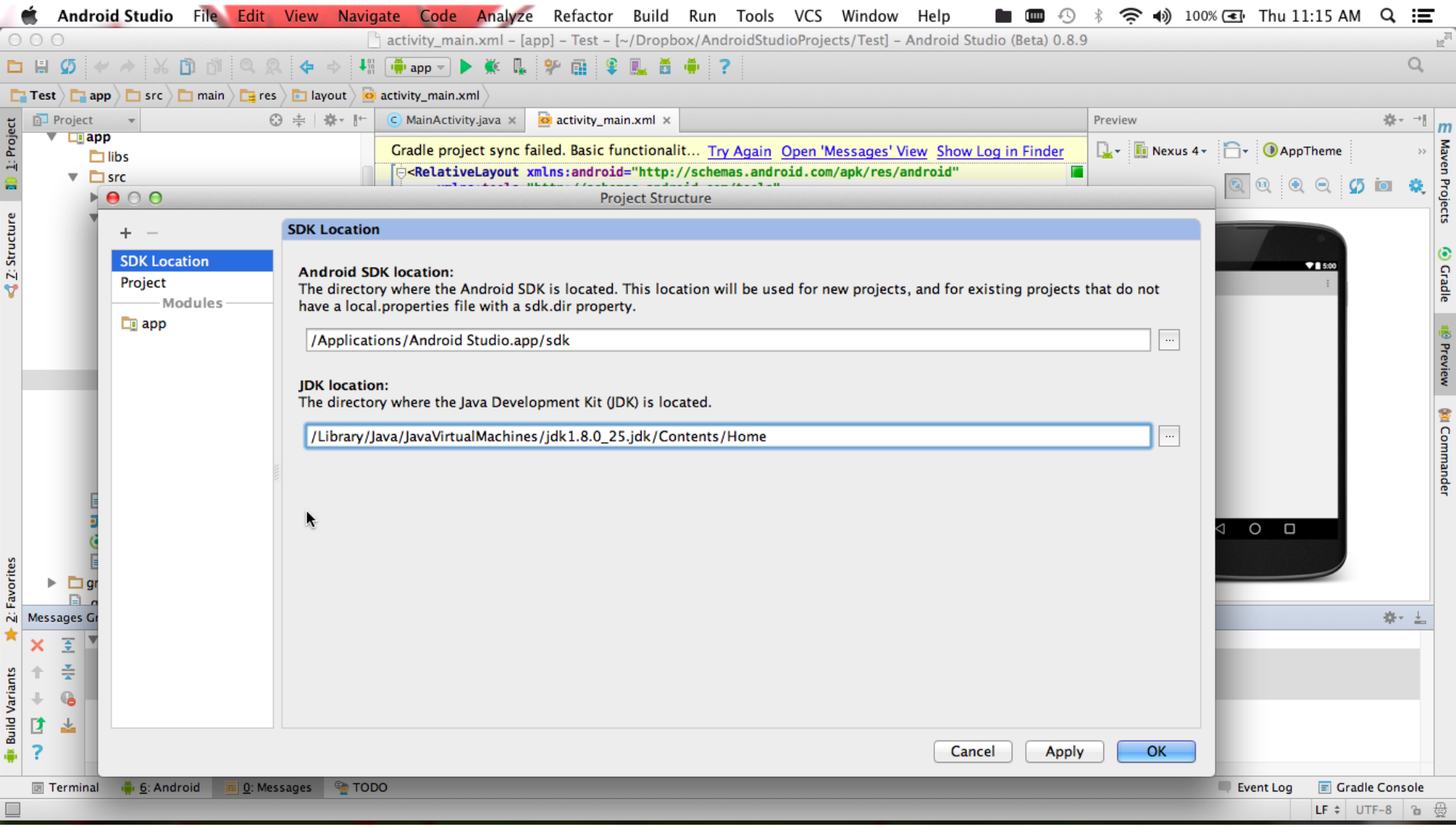


Determine New JDK Location



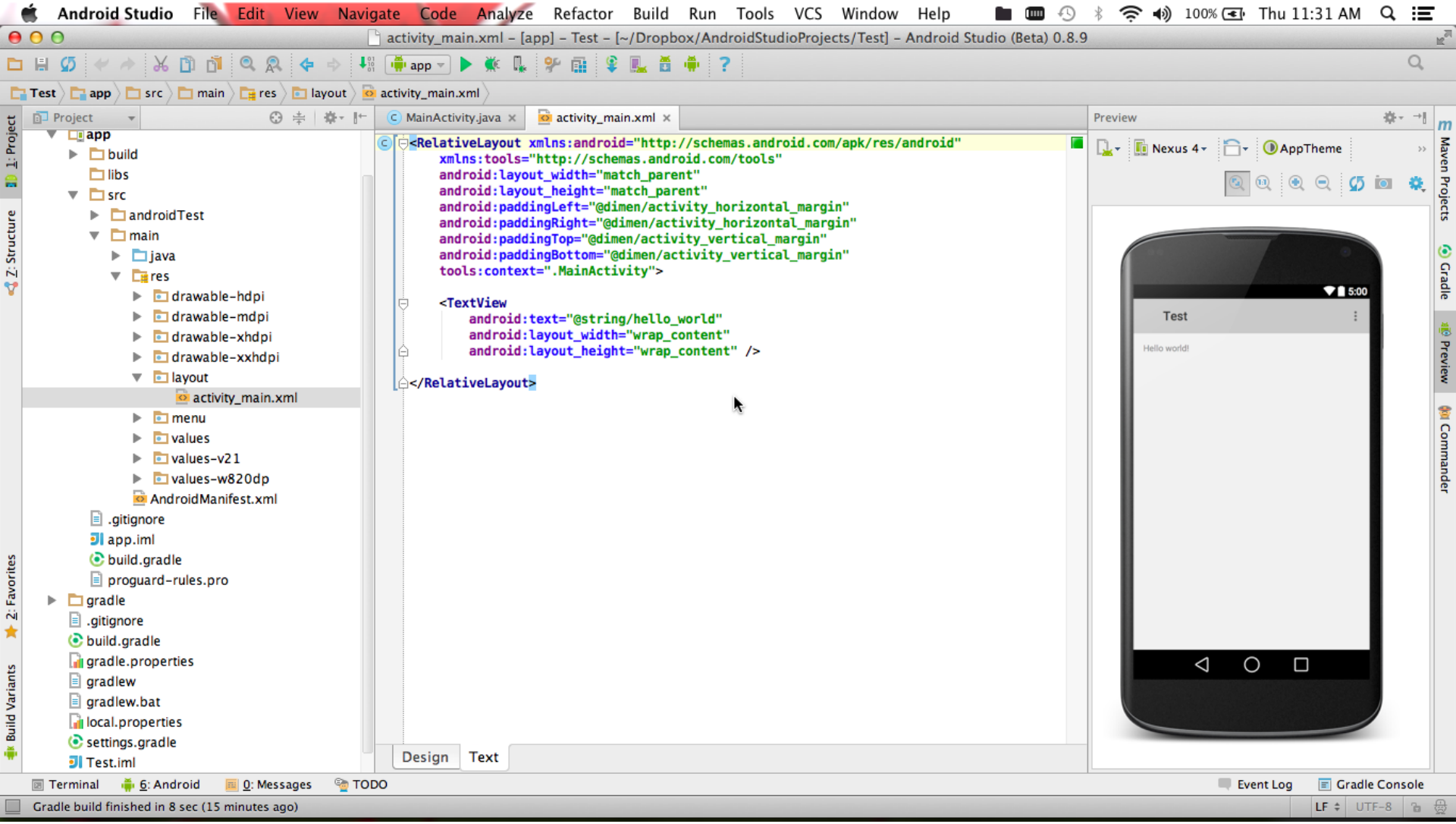
The weird thing on the Mac is that JDK6 is installed in one place and JDK8 is installed in another (see terminal screenshot above). It's likely that your JDK7 and/or 8 are installed at /Library/Java/JavaVirtualMachines

Enter in New JDK Location



Enter in this new JDK location. Once you've done this, click 'OK.' The Gradle sync process will likely auto-restart. If it doesn't, press "Try Again."

WHEW, FIXED!

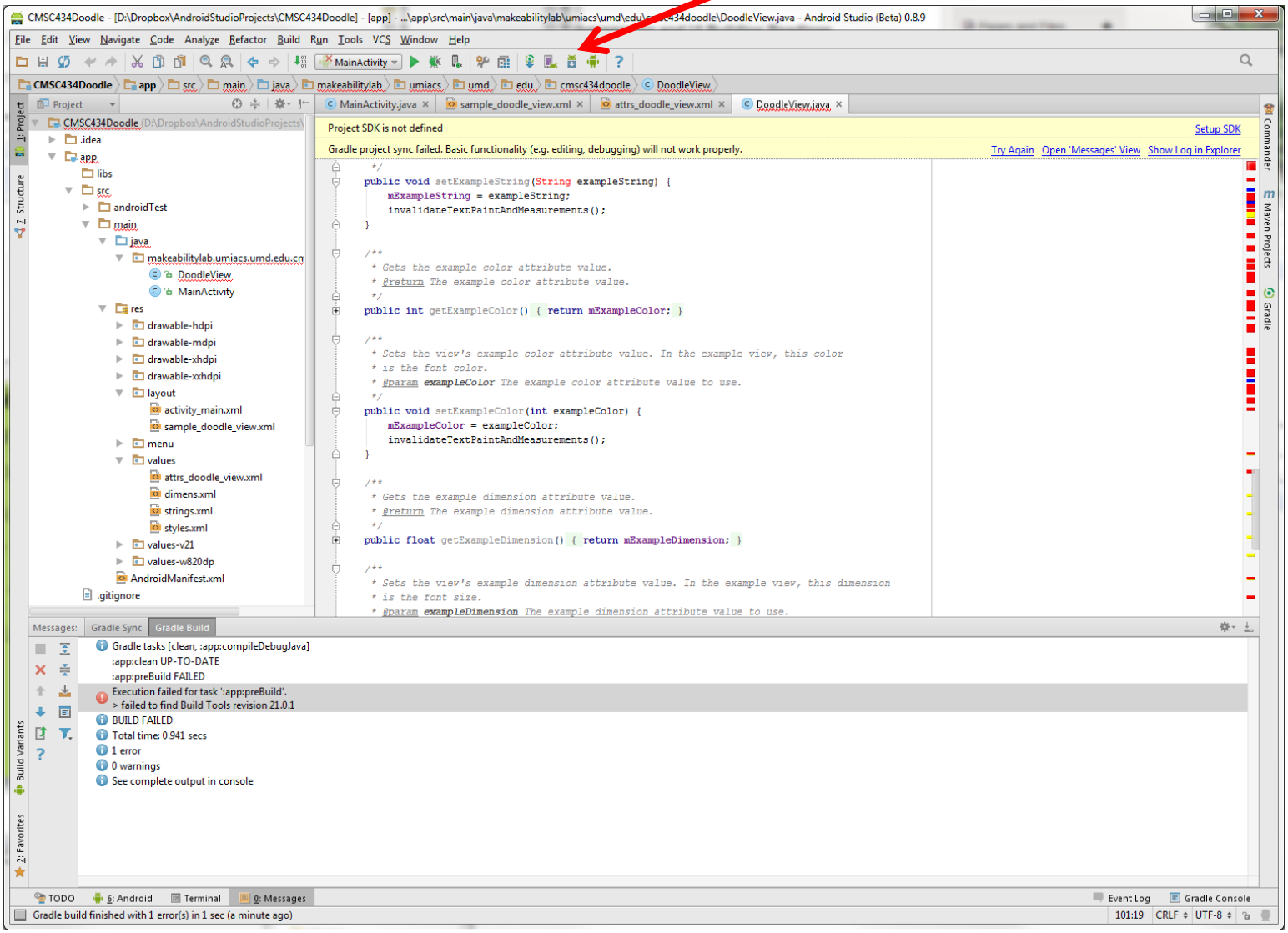


If only I'd experienced this problem before, I could have been prepared to solve it in class!

Failed to Find Build Tools

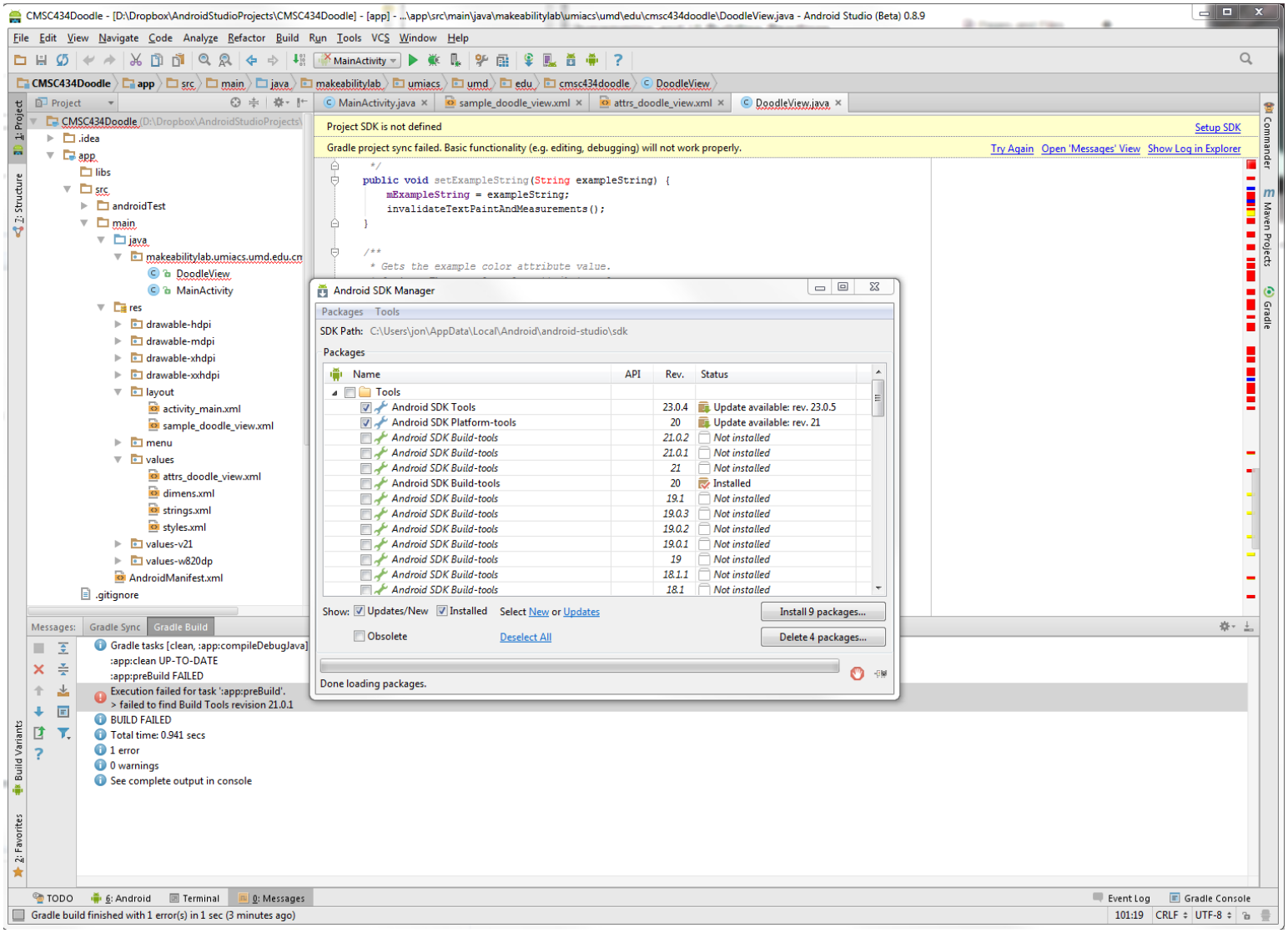
Failed to Find Build Tools

Click on the SDK Manager



Here, I just loaded an existing project on a new computer that doesn't have the proper build tools version installed. So, click on SDK Manager and select Build Tools revision 21.0.1

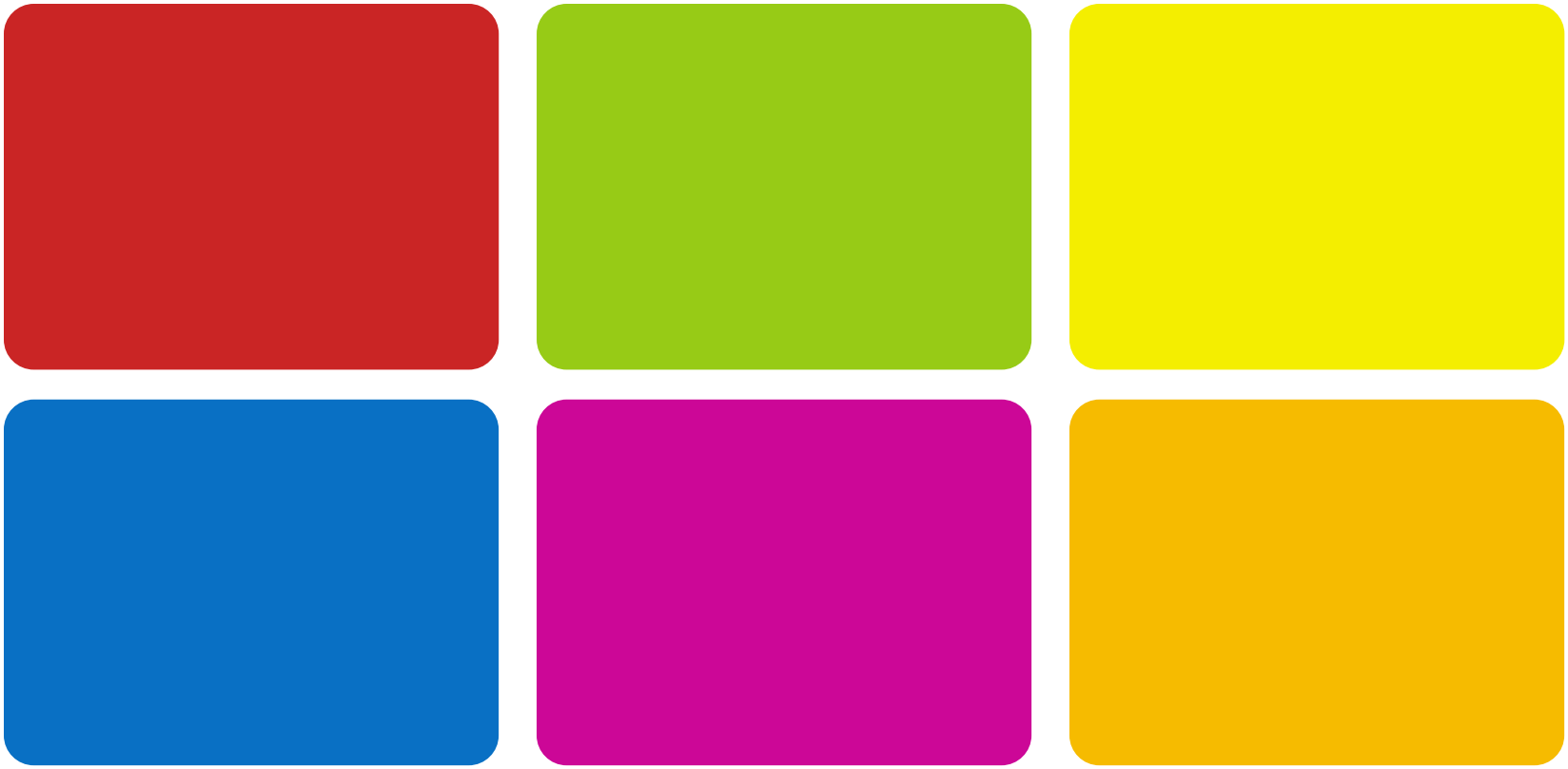
Install Missing Build Tools Version



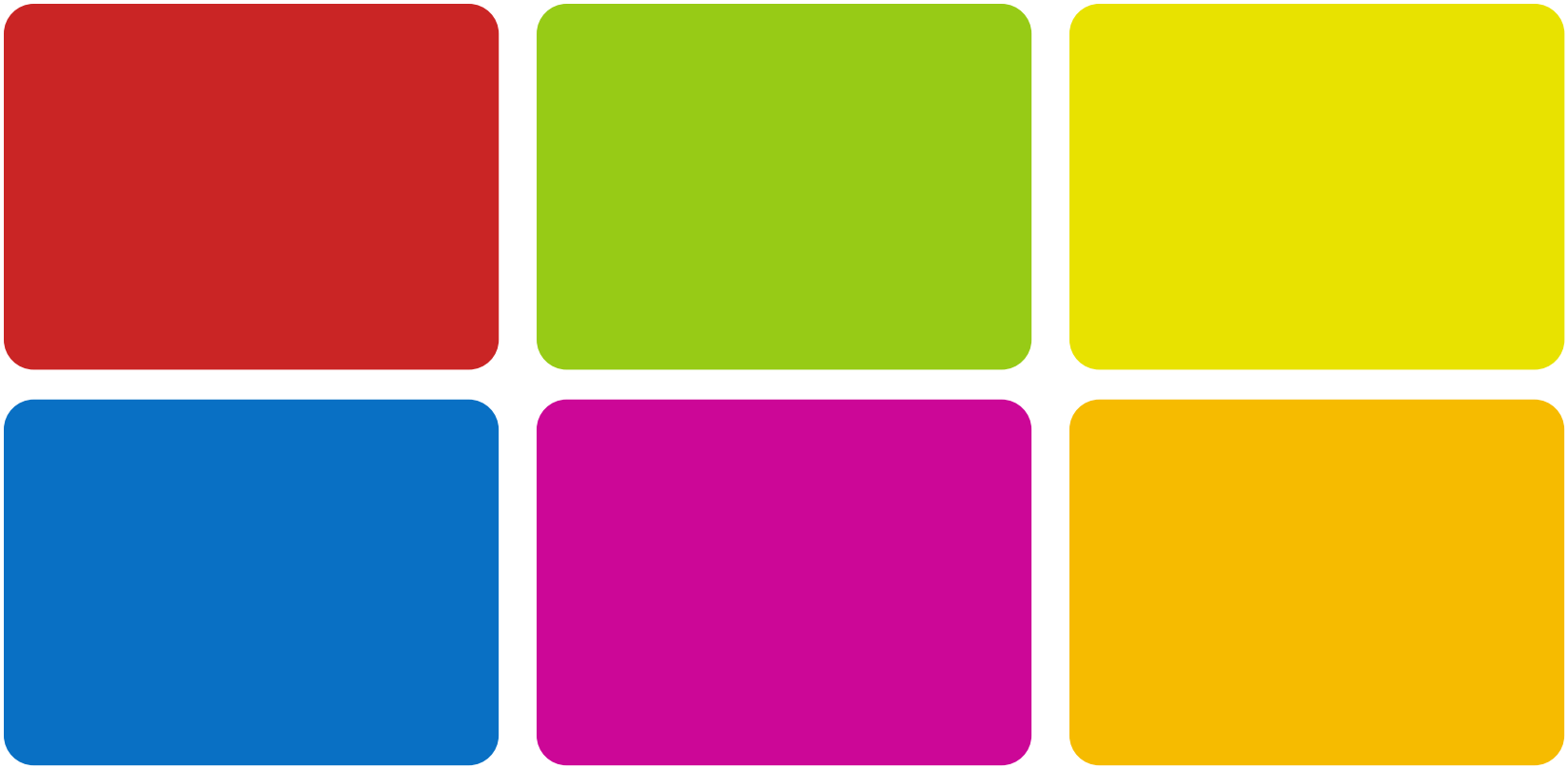
Click on and install Android SDK Build-tools 21.0.1

Find Your Project Team

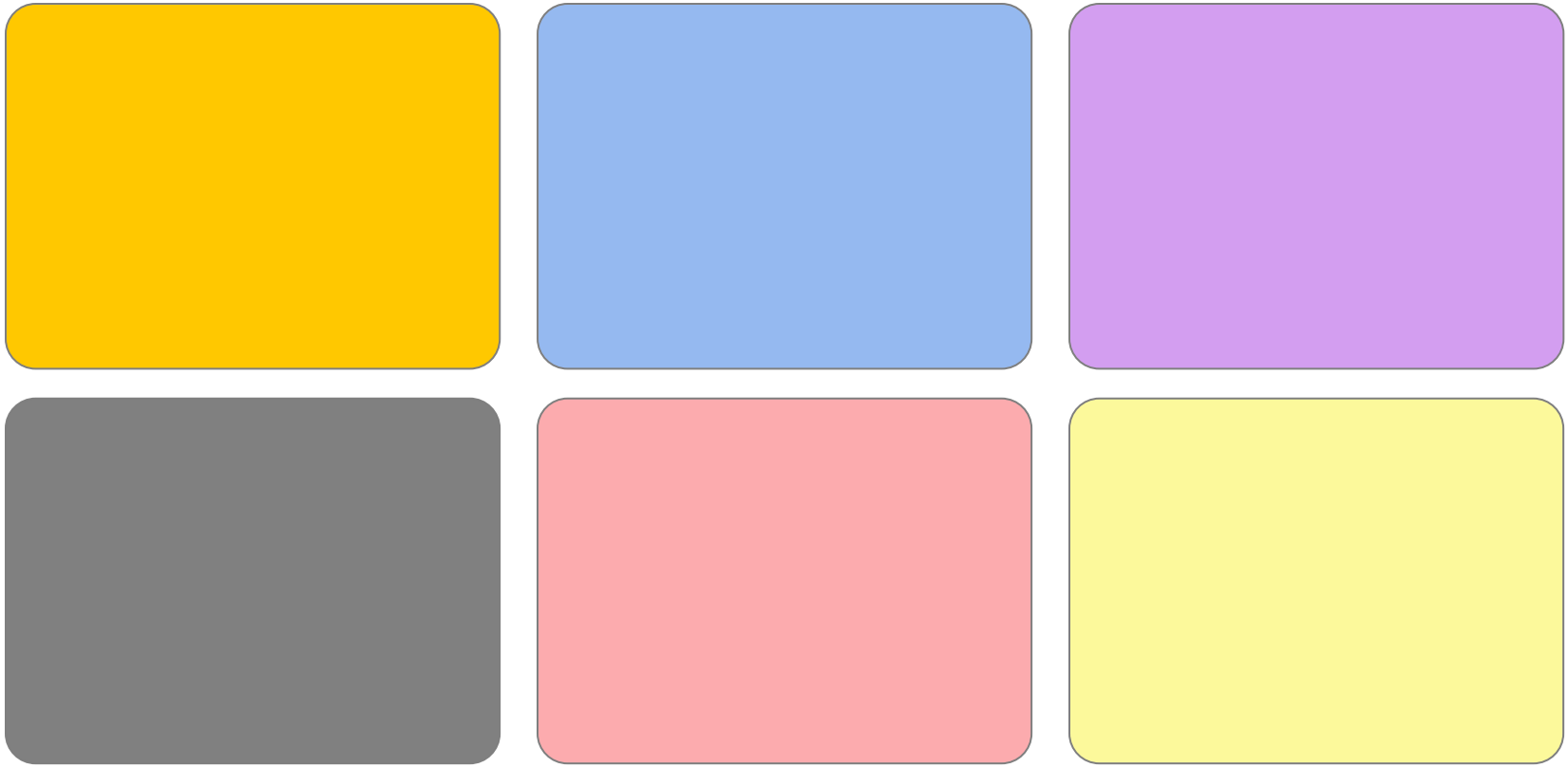
Check-in on TA05



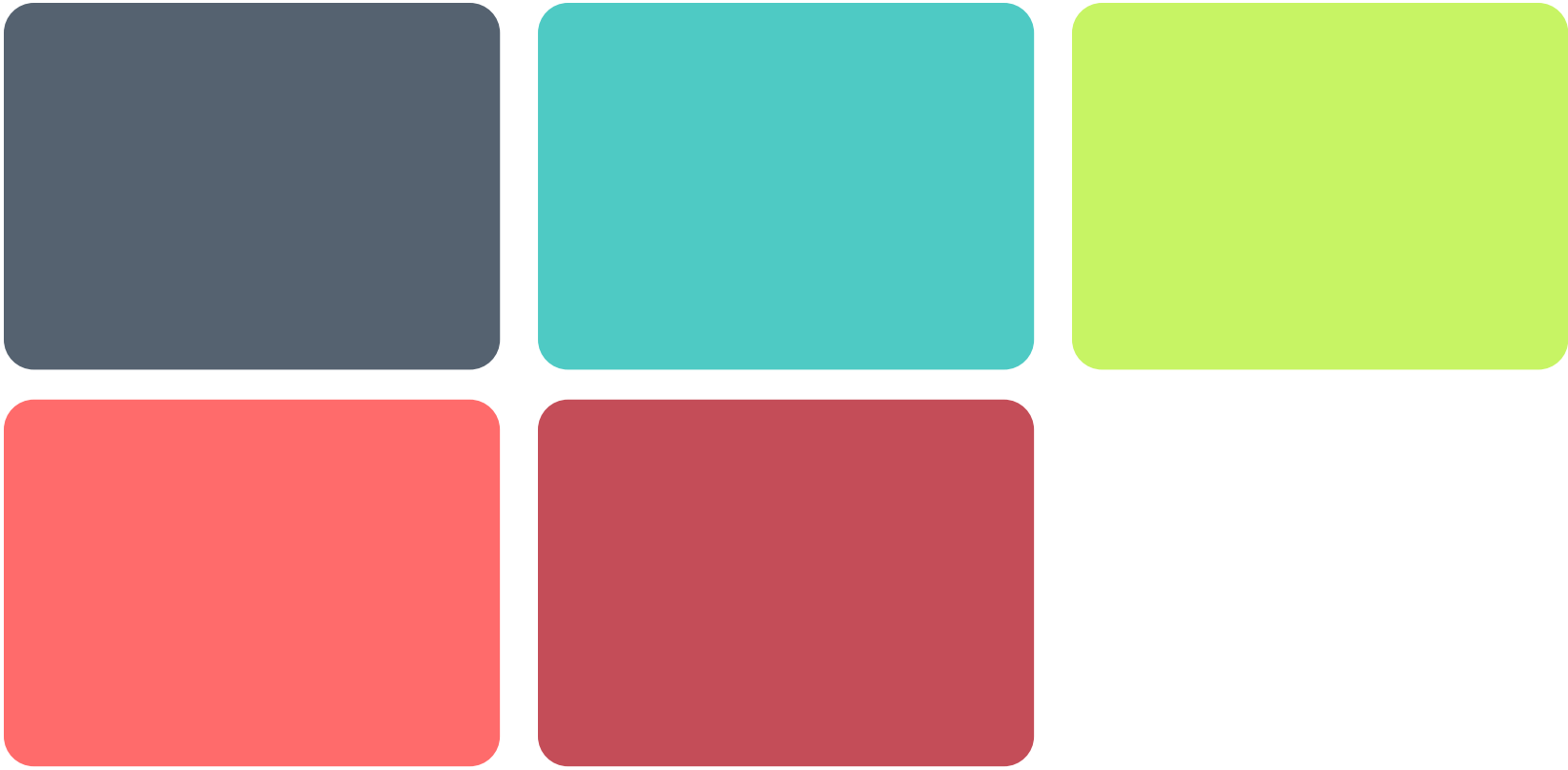
Dark Palette



Light Palette



Smartsheet Gantt Palette



Light Palette