Review Material for Lesson 1

Congratulations on finishing lesson 1! You've come a long way already: you've created your first app, and have learned many of the essential Android concepts. We studied the tools used to create apps, including Android Studio, the Android SDK, testing devices (both real and emulated), and the Gradle build system. We also looked at some of the core classes that make up Android apps, including activities, fragments, layouts, views, list views, and adapters.

New Concepts

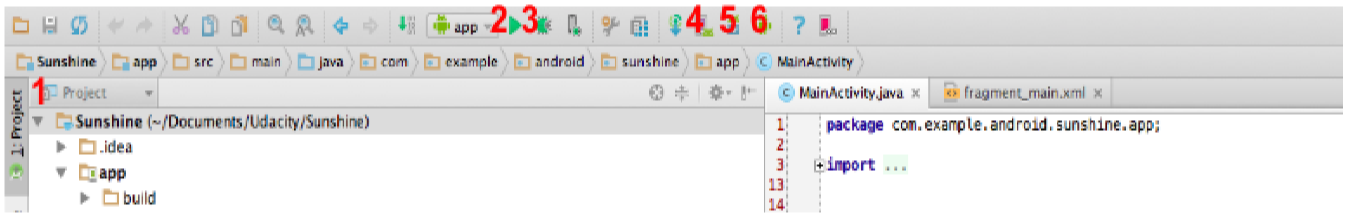
- Android Studio
- SDK - Target and Minimum
- Emulators vs. Real Devices
- Gradle
- Application
- Activity
- Fragment
- Views and ViewGroups
- Views and XML layouts
- ListView
- Adapter

Android Studio

Android Studio is the IDE that we're using for the course. When you start a new app, you will create a **Project** in Android Studio. Every project will contain one or more **modules** for each logical part of your application. To ensure that your module name is unique, the module naming convention is to use the reverse of your internet domain name.

Android Studio organizes your project in a specific way, described here.

The following is a review of some of the important buttons in Android Studio.



```
package com.example.android.sunshine.app;

import ...
```

1. This determines how the project structure will be shown. In the course we use the **Project** view but the**Android** view is also a handy way to organize your files.

2. This is the **run** button. It will save and compile the current state of your project as an apk and then launch it on either an Android phone connected to your computer or an emulator.

3. This is the **debug** button. It runs your code as above, but attaches the Android debugger, which allows you to do things like stop at breakpoints and step through the code.

4. This is the **AVD manager** button. It opens the Android Virtual Device manager, which allows you to create, delete, run and edit Android emulators.

5. This is the **SDK Manager** button. The SDK Manager allows you to download new SDKs for different Android APIs.

6. This is the **Android Device Monitor** button. The Android Device Monitor shows you a lot of information about your emulated and attached devices. For example, you can do basic performance analysis and see what’s on the file system.

SDK - Target and Minimum

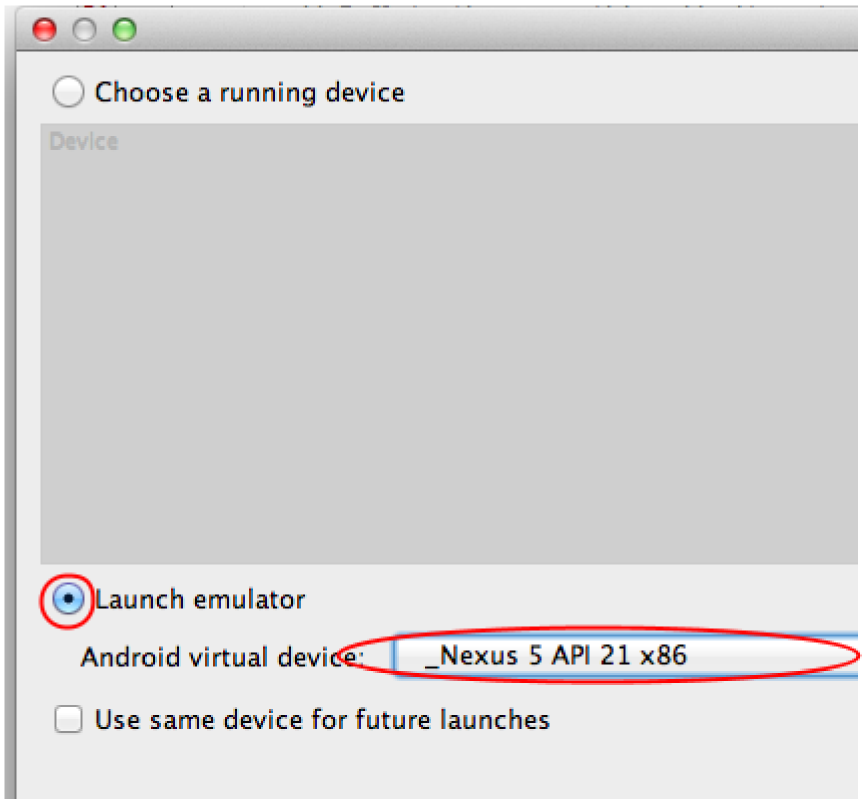
When you create a new project, you need to define a minimum SDK, or Software Development Kit. The SDK is a set of tools that you download that allows you to compile and create Android Projects. SDKs match API levels, when a new API level is released, a new SDK is released to make projects for that level. A current SDK is automatically downloaded when you install Android Studio.

The minimum SDK is the minimum Android API level that the application is meant to support. In our case we are supporting API level 10, which includes all devices running Gingerbread and beyond. This will support 99% of devices on the market.

The target SDK, which is automatically defined for you, signifies the SDK that you use to compile and test with. It should be the most current SDK.

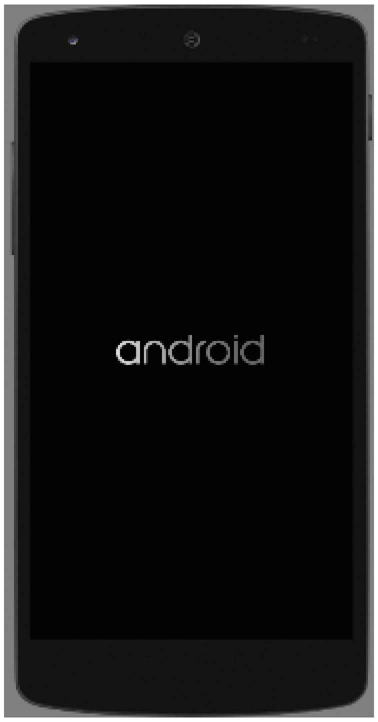
Emulators vs. Real Devices

When you download Android Studio 1.0 it comes with an emulated phone. You can start this emulated phone on your computer by simply running your app and choosing the “Launch emulator" option and picking your emulator:



Launch emulator
Android virtual device: _Nexus 5 API 21 x86
Use same device for future launches

Note, sometimes emulated phones hang with a black screen or loading screen like this one:



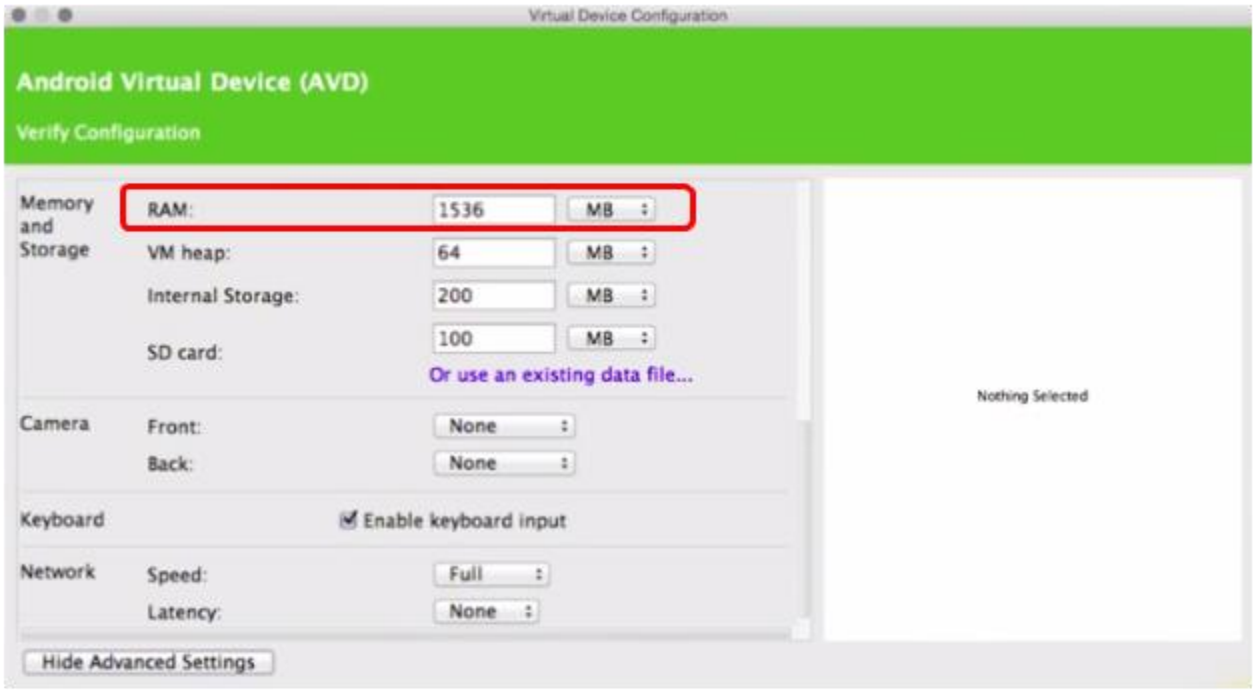
If this happens to you, check out your error messages in Android Studio. A common message is:

```
/Android/sdk/tools/emulator -avd _Nexus_5_API_21_x86 -netspeed full -netdelay none

emulator: The memory needed by this VM exceeds the driver limit.

HAX is not working and emulator runs in emulation mode
```

This means that your machine is not fast enough to run the default emulator. Not to worry; follow Katherine's instructions to make a new emulator and reduce the RAM that the emulator uses. Here is the configuration screen where you can do this:


Virtual Device Configuration

Android Virtual Device (AVD)

Verify Configuration

Memory and Storage	RAM:	1536	MB
	VM heap:	64	MB
	Internal Storage:	200	MB
	SD card:	100	MB
		Or use an existing data file...	
Camera	Front:	None	
	Back:	None	
Keyboard		✓ Enable keyboard input	
Network	Speed:	Full	
	Latency:	None	

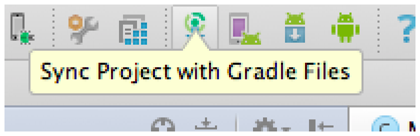
Emulators are great, but they can be a bit slow. If at all possible, we suggest you use a real device. You'll need to follow the steps in this video to unlock developer mode, but ultimately the performance will be much faster and experience much smoother.

If you're set on using the emulator, consider installing Genymotion. The steps to do so can be found here.

Gradle

Gradle is the build system that packages up and compiles Android Apps. Android Studio automatically generates Gradle files for your application, including the `build.gradle` for your app and module and the `settings.gradle` for your app. You do not need to create these files. You can run Gradle from a terminal (described in this and this video), but you can also use the *run* button which will automatically run the Gradle scripts in your project. For more information on the build process that Android Studio and Gradle are automatically handling for you, checkout the developer guide.

TIP: if your project is having Gradle issues, sometimes clicking the **Sync Project with Gradle Files** button helps. Running clean and rebuilding your project can also help resolve errors.



Application

You probably know what an Android application, like Gmail or Keep, looks like as a user, but what does an application look like from a developer's perspective? An application is a loose collection of classes for the user to interact with. The UI components are organized into Activities, which we learned about in this lesson. The behind-the-scenes work is handled by other Android classes including:

- Content Providers (Lesson 4) - Manage app data.
- Services (Lesson 6) - Run background tasks with no UI, such as downloading information or playing music.
- Broadcast Receivers (Lesson 6) - Listen for and respond to system announcements, such as the screen being turned on or losing network connectivity.

Activity

Activities are the components of Android apps that the user interacts with and a core class in Android. When you create an app with Android Studio, it will create an initial activity class that will start when the app is launched. The default name of this activity is `MainActivity`. An activity is a single, focused thing that the user can do and roughly maps to one screen of the app.

Fragment

Activities can contain one or more Fragments. Fragments are modular sections of an activity, usually meant to display UI. Two activities can have the same fragment and fragments can be added or removed from an Activity. An Activity with blank Fragment is what we created for Sunshine. The `PlaceholderFragment` is automatically generated as an inner class of the activity, but fragments don't *need* to be inner classes.

Views and ViewGroups

A view is the basic building block for user interface components. A fragment might combine multiple views to define its' layout. Buttons, text and other widgets are subclasses of views and can be combined in ViewGroups to create larger layouts. Common ViewGroups include:

- `LinearLayout` - For horizontal or vertical collections of elements.
- `RelativeLayout` - For laying out elements relative to one another.

- `FrameLayout` - For a single view.

Since views nest within other views, this creates a tree like structure of views for every layout.

Views and XML Layouts

To describe our user interface, we describe layouts using XML. The layout defines a collection of views, view groups and the relationships between them. Our layouts are stored in the app/src/main/res/layout directory. To turn an xml layout into java view objects, we need to **inflate** the layout. After the layout is inflated, we need to associate it with an Activity or Fragment. This process of inflating and associating is a little different depending on whether it's a layout for an Activity or Fragment.

For an Activity

We inflate the layout and associate it with the Activity by calling the `setContentView` method in `onCreate` in our Activity:

```
setContentView(R.layout.activity_main);
```

For a Fragment

In our Fragment classes we inflate the layout in the `onCreateView` method, which includes a `LayoutInflater` as a parameter:

```
View rootView = inflater.inflate(R.layout.fragment_main, container, false);
```

The root view, or view element which contains all the other views, is returned by the `inflate` method of the `LayoutInflater`. We then should return this rootView for the `onCreateView`.

ListView

ListView is a subclass of View optimized for displaying lists by displaying many copies of a single layout. We are going to use a ListView to display our weather information in Sunshine. Each row of weather information is defined by a layout called `list_item_forecast.xml`. The list view contains multiple copies of list_item_forecast.xml, one for each row of weather data.

An `Adapter` is used to populate a ListView.

Adapter

Adapters translate a data source into views for a ListView to display. In our case we used an ArrayAdapter to take an array as our data source and populate our ListView with the data from the array. Here's the corresponding video about how this process works.

Instructor Notes

Welcome to *Developing Android Apps: Android Fundamentals*, and welcome to the course Instructor Notes! You can often find useful links, documentation, and other help down here!

Reto talks a lot about what we *won't* be covering, but don't worry - Dan and Katherine will make sure we include everything you need to know to get started and become productive Android developers as quickly as possible.

You *will* need to do some of your own research throughout the course, so be sure to check out some of the developer resources Dan mentions, including:

Android Sample Code
Android Developer Site
Android Developers on Google+
Android Developers on Twitter
Android Questions on Stack Overflow

You don't have to take Katherine's word for it - you can see some of her impressive coding abilities in action by downloading Google Keep from Google Play. It's got 4.5 stars and over 10 million downloads, so you know she's legit.

Dan talks a lot about his hats, but don't let that distract you from the equally impressive variety of facial hair he's worn during his time at Google.

Reto really has been around since the very early days of Android, writing this historical getting started guide a week after the first beta Android SDK was released. Despite appearances, he's not really a scientist: he just really likes wearing that lab coat (it makes him feel more serious).

Find the Android Studio Installation and Setup Guides for Windows and Mac in our How to Install Android Studio mini-course.

Setup Java and Android Studio before following Katherine's instructions to create the project.

Don't forget you can check out Android Studio installation instructions on the developer site and the Android Studio known issues if you're having trouble translating what's in the video to what's in your IDE

Pie charts *are* awesome, and with 1.5 million new Android devices being activated daily, the distribution of platform versions on active Android devices changes frequently.

The graph shown in this video is already out of date - check out theAndroid Platform Version Distribution dashboard to get the latest distributions. The same page also provides the relative proportion of devices with different screen sizes and pixel densities.

Learn more about the specific features released in each version of Android by reviewing the Android version history, and find out how that history is depicted by way of life-size food sculptures in *Story time with Reto* at the end of this lesson.

And in case you couldn't read the future releases Reto suggests for L and M (around 1:40 in the video), that's "Lamington?" and "Meringue?"

The target SDK for your app should *always* be the latest Android platform you've tested against.

In order to provide a great experience for as many users as possible, use the Support Library to use features from newer platform releases on devices running older versions and review the Android Platform Version Distribution dashboard to get the latest platform distribution numbers.

You can download the placeholder launcher icon (right-click and save) to use as our app icon, until we create a *real* logo for Sunshine. We go over the way to add an app icon here.

When creating the project, make sure to select **Blank Activity With Fragment**.

Check out the Android Studio installation instructions and the Android Studio known issues if you're having trouble translating what's in the video to what's in your IDE.

You can download the placeholder launcher icon (right-click and save) to use as our app icon, until we create a *real* logo for Sunshine.

Once again, notice that the Android Studio *New Project* wizard has changed since filming. In the latest release you may not have to select a Target SDK yourself -- that's because there's never any reason to create a new project not targeting the newest platform. Keep in mind that you'll want to increase the target SDK for your app when a new version of the Android SDK becomes available. We go over the way to add an app iconhere.

Check out the Android Studio installation instructions and the Android Studio known issues if you're having trouble translating what's in the video to what's in your IDE.

A Note about Updating to AndroidStudio 1.0 and The GitHub Repository

Throughout the course, the code changes seen in the videos are stored on branches in this GitHub repository. For example, the first state of the official Sunshine app can be found at branch 1.01-hello-world. 1.01 refers to lesson 1, step 1, where as branch 4.12 is lesson 4, branch 12.

With the updates to Android Studio 1.0, some of the gradle files are broken. If you would like to import a GitHub branch of the repo or update your own previous version of the code to be compatible with 1.0, make sure your build files contain the following:

For the `build.gradle` in the **project** folder:

- classpath should be 'com.android.tools.build:gradle:1.0.0'

For the `build.gradle` in the **app** folder:

- replace runProguard with minifyEnabled
- Set compileSdkVersion to 21
- Set targetSdkVersion to 21

The Android Developer site has more details for Managing Virtual Devicesand using the Android Debug Monitor to control emulators.

Virtual devices offer a great way to test and debug your app on a wide variety of devices, but if you own a *real* Android device, it's often faster and easier to use for your initial development.

Throughout this course, we recommend using a real Android device running Ice Cream Sandwich (Android 4.0+) or higher. If you don't have such a device, use the Nexus 5 virtual device you created in this lesson. Check out the documentation on Creating an AVD and New Project under the course resources page for more detailed guidance.

Gradle is a sophisticated build system that lets you use a simple, declarative DSL to configure the build process of your app. For more info, check out Udacity's Gradle for Android and Java course!

You can find out more about the Android build system here, or by watching Xavier Ducrohet describe the new Android SDK build system at Google I/O 2013.

If you're a hardcore command-line guru, be sure to read up on theAndroid Debug Bridge (ADB) and other command line tools. You can also find out more about the Android Virtual Machines ART and DART to see how the Android Runtime works under the covers.

You may need to run chmod +x on gradlew before you can run it. See thislink for more detailed instructions.

Setting up USB debugging on some operating systems can be a little more complicated than Katherine demonstrated - check out this guide on Using Hardware Devices for debugging to learn more if you run into trouble.

Success Kid's response to this video

Command Line Tool Commands

The usage of these commands is entirely optional. The result is that same as clicking the Run button in Android Studio.

- `chmod +x gradlew` - This command only needs to be run once and is used to give gradlew the correct execute permissions.
- `./gradlew assembleDebug` - This command will compile the code.
- `adb install -r app/build/outputs/apk/app-debug-unaligned.apk` - This command will install the APK. With the `-r` flag it will overwrite any prior installed versions. **Note** if you have more than one device, you will need to use the `-s` flag right after adb to specify the serial number of the intended device.
- `adb shell am start -n com.example.android.sunshine.app/com.example.android.sunshine.app.MainActivity`- This command will actually run the app.

If you're all about the command-line, check out these links to find out more about using the Android Debug Bridge (ADB) and building with Gradle.

ProTip: You can select multiple devices from the device picker!

PlaceholderFragment is embed in the MainActivity class. You are probably using a more recent version of Android Studio that creates a separate Java file called MainActivityFragment.

You'll also notice that Dan is defining our entire UI within XML files. These XML files are then "inflated" into live Java View objects. It's also possible to create your layout in code at runtime, but using XML is considered best practice - and has many significant advantages you'll learn about throughout the course.

The Activity class that Dan's extending with MainActivity is the screen, or window, that's used to display a visual user interface to our users. You'll learn more about Activities in Android in Lesson 3.

The Fragment class extended with PlaceHolderFragment is used to encapsulate portions of an Activity within the UI. You'll find out why Fragments are awesome in Lesson 5.

The Visual Layout Editor is a powerful tool for creating layouts. You can find out more tips on using it in this guide to using the Layout Editor.

You're testing an app on your favorite phone device, and everything looks great, but then you rotate it into landscape, or run the same app on a larger phone or a tablet, and *it's the same UI just squeezed into the top left corner* and you're all like...

To learn more about Responsive Design in Android, check out some of Google's Android Design Advocates as they discuss various elements of Android Design in this episode of Android Design in Action.

They look at the apps including Android Calendar, Pattrn, Pocket, TED, and the 2012 Google I/O app and discuss how they respond to differing device form factors.

You can find lots more Android design tips from the full Android Design in Action playlist.

We'll be exploring each of the Layout Managers in more detail as they're used throughout the course -- but if you want to ~~cheat~~ read ahead, you should read these developer guides on using Linear Layout and Relative Layout.

As Dan mentioned, each of these Layout Managers extends the View Group class - itself an extension of the View class - designed specifically to contain and layout multiple child views. That means you could, if you were feeling particularly adventurous, create your own Layout Managers.

---

the Scroll View here - an extension of the FrameLayout class (meaning you should place one child in it containing the entire contents to scroll) that allows the user to scroll vertically to reveal more content than can be displayed on screen at once.

As indicated in this example, the ScrollView is often used to contain a vertically oriented LinearLayout.

A ScrollView is layout container that can be scrolled by the user, allowing its contents to be larger than the physical display. The simplest way to implement a list of 50 items is using a LinearLayout containing all 50 items and and placing it within a ScrollView that will let you scroll to display different items.

But creating all 50 Views is expensive. The purpose of this quiz was to get you thinking about an alternative that uses fewer Views. What's the minimum number of inflated Views you'd need?

The ListView is a specialized control that is optimized for displaying long lists of items. It's very efficient in how it creates, recycles, and displays views, and is optimized to scroll very smoothly. It's also designed to simplify the process of adding, removing, and editing items that are to be displayed in the list -- as you'll learn later in this lesson.

We'll explore the ListView in more detail throughout the course, but if you really want to understand some of what's possible with this control, you should watch The World of ListView from Google I/O 2010.

Adapters are the glue that allows us to bind our underlying data to our user interface elements.

Check out the Android API guide on AdapterViews.

Adapters are a little unusual compared to similar mechanisms in other frameworks, in that the Adapter itself is responsible for creating the Views that are displayed within the bound AdapterView (for example a ListView) - while the AdapterView is responsible for how those views are laid out.

In our case, we're going to use an ArrayAdapter to bind the ArrayList we just created to the ListView we added to our layout earlier. We'll learn more about building layouts with an adapter as we continue creating Sunshine.

---

an ArrayAdapter, an implementation of the Adapter class, designed to be used when the underlying data is stored in an ArrayList. There are alternative Adapter implementations for other kinds of underlying data -- specifically Cursors -- that we'll investigate in later lessons.

You can see an example of an ArrayAdapter in the Android documentation.

This is the first time you've encountered the magic R files. It's not really magic. Check out this guide for more info!

Because we're creating all our layouts in XML, it's important to understand how you can get a reference to those Views within your code. That means you'll be making a lot of calls to findViewById.

Notice how Katherine tried to slide that last part by you? If you're having trouble assigning the ArrayAdapter to the ListView within the layout, check out this example of an ArrayAdapter in the Android documentation for help.

Note that findViewById is both a method on the Activity class, and on theView class.

Now that you've gotten your feet wet in Android Studio, watch this Webcast to learn Tips, Ticks and Troubleshooting techniques.

Associated handout here.