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

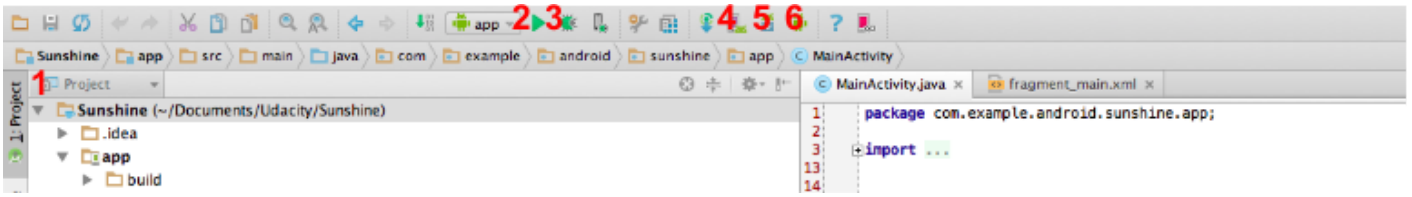
- Android Studio
- SDK - Target and Minimum
- Emulators vs. Real Devices
- Gradle
- Application
- Activity
- Fragment
- Views and ViewGroups
- Views and XML layouts
- ListView
- Adapter

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.



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

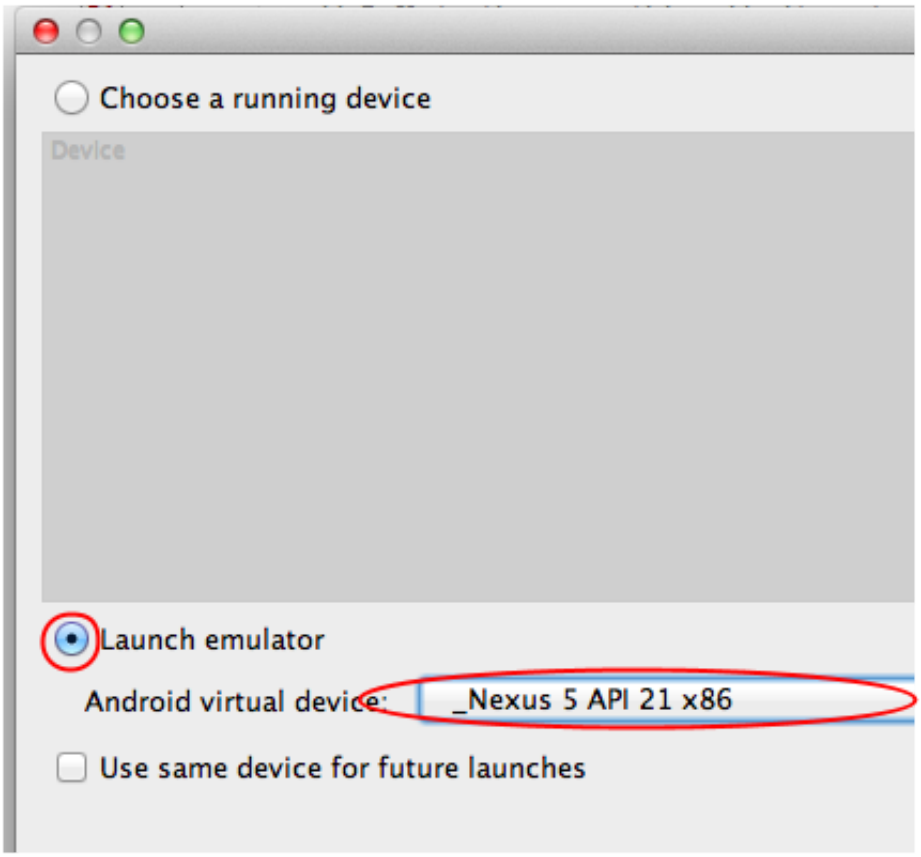
When you create a new project, you need to define a minimum SDK, or Software Development Kit. The SDK is a set of tools that you download that allows you to compile and create Android Projects. SDKs match API levels, when a new API level is released, a new SDK is released to make projects for that level.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

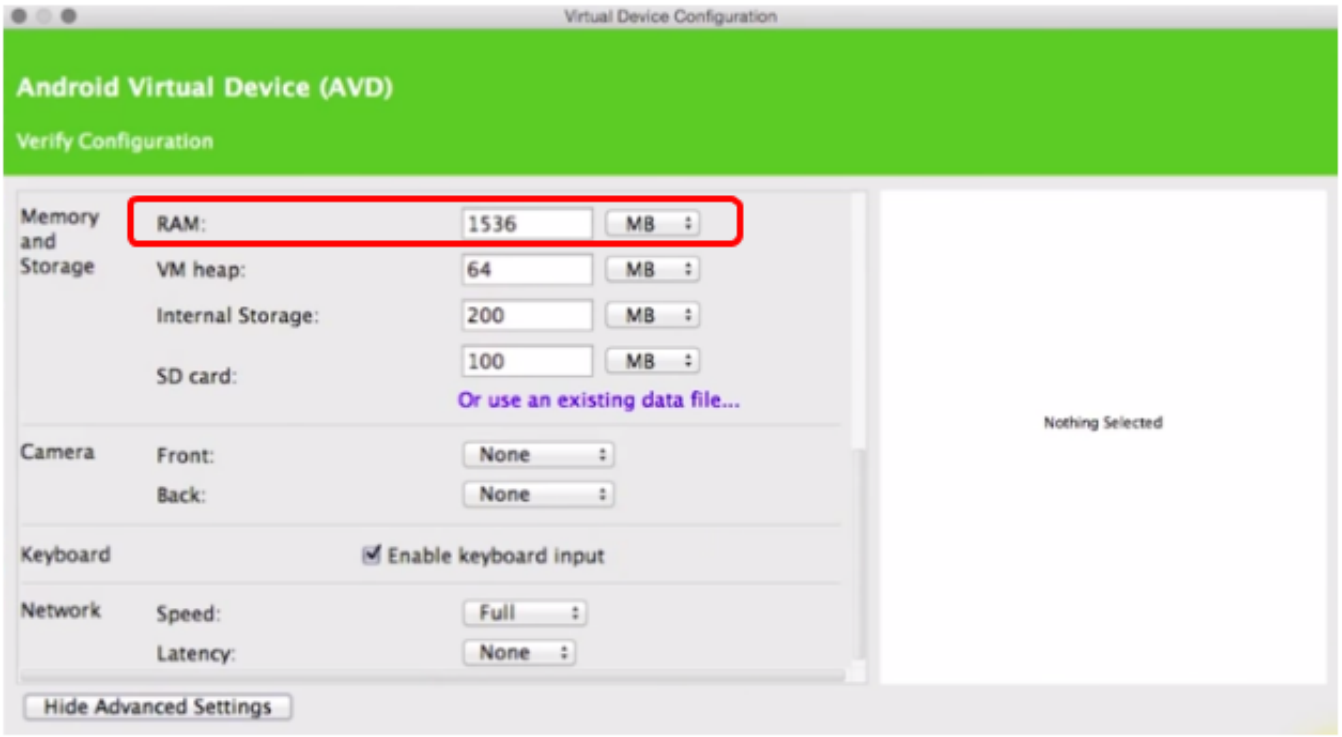


Note, sometimes emulated phones hang with a black screen or loading screen like this one:

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:



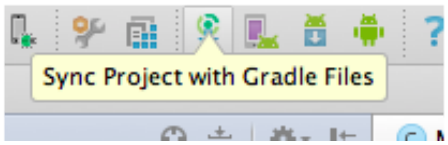
Emulators are great, but they can be a bit slow. If at all possible, we suggest you use a real device. You'll need to follow the steps in this video to unlock developer mode, but ultimately the performance will be much faster and experience much smoother.

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.