

Mobile Application Development

Week 8: Android Development Overview

Android

The Android operating system was originally developed by Android Inc which Google purchased in 2005.

Android 1.0, the first commercial version, was released on 9/23/08.

It now has 75% of the worldwide market share compared with 25% for iOS. (slide)

In the US iOS has 58% and Android 41%. (slide)

Android Development

Developer web site <http://developer.android.com>

Android Studio (or Download link)

<http://developer.android.com/studio/> (scroll down for system requirements)

- Android development can be done on Windows, Mac OS, or Linux systems
 - Note that MacOS 10.15 Catalina is STILL not listed as supported. Oh well.
- Android Studio
 - Integrated Development Environment (IDE)
 - Android Software Developer's Kit (SDK)
 - Emulator to run, test, and debug your apps on different virtual devices
 - Tools and profilers for testing
 - Initially released in 2013, previously the SDK was bundled with Eclipse
 - Can also use the SDK with the command line or another SDK
- Java programming language
 - Although Android uses Java it does not run .class or .jar files, it uses its own format for compiled code
- Kotlin programming language
 - Kotlin was developed by JetBrains in July 2011
 - In February 2012 Kotlin was open sourced
 - Kotlin 1.0 was released in February 2016
 - Google announced Kotlin support at Google IO in 5/17
 - Integrated into Android Studio 3.0
 - Kotlin-first <https://developer.android.com/kotlin/first>
 - In May 2019 Google announced that Kotlin was now its preferred language for Android development.
 - Kotlin is cross platform and fully interoperates with Java
- eXtensible Markup Language (XML)

Use the SDK Manager (Tools | SDK Manager) to download additional tools and components.

Android Setup

The first time you launch Android Studio it will take you to a setup wizard.

Configure | SDK Manager to install other needed SDK components.

OR

Tools | Android | SDK Manager to install other needed SDK components.

SDK Platforms (show package details)

- Android 11.0(R) (API 30)
- Show Package Details
 - Android SDK Platform

- Sources for Android
- Google APIs Intel x86 Atom System Image (for the emulator)

SDK Tools

- Android SDK Build tools
- Android Emulator
- Android SDK Platform-Tools
- Android SDK Tools
- Documentation (do this later)
- Intel x86 Emulator Accelerator (HAXM Installer) (do this later)
- Layout Inspector image server for API 29-10 (do this later)

SDK Update Sites

All should be checked.

Chose top license, click Accept. Install.

Android EcoSystem

- Google -> phone manufacturers (OEMs)
- OEMS -> carriers
- Android updates take about 3-6 months for a new version is available on a carrier
- Android versions – version, dessert name, API
 - When creating a new project you need to decide what Android versions to support

Android Devices

Android runs on devices that are all different

- Screen sizes
- Processor speed
- Screen density
- The number of simultaneous touches the touch screen can register
- The quantity and positioning of front and back cameras
- Bluetooth

Screen densities https://developer.android.com/guide/practices/screens_support.html

Pixel densities <https://developer.android.com/training/multiscreen/screendensities>

The density-independent pixel, dp, is an abstract unit based on the physical pixel density of the screen. 1 dp is equal to 1 physical pixel on a medium density screen (160 dpi screen).

At runtime, Android automatically handles the coversion of dp units based on the density of the screen.

The conversion of dp units to screen pixels is: $px = dp * (dpi / 160)$.

So on a 480 dpi screen, 1 dp equals 3 physical pixels.

Android Terminology

- An activity is a single, defined thing a user can do
 - Usually associated with one screen
 - Written in Java or Kotlin
 - Android Studio has activity templates
- A layout describes the appearance of the screen
 - Design view
 - Written in XML
- The emulator lets you define Android Virtual Devices (AVD)

- A device configuration that models a specific device
- Simulators imitate the software environment of the device but not the hardware.
 - They have access to all the host (Mac) hardware's resources
- Emulators imitate the software AND hardware environments of the actual device
 - A flight simulator simulates a flight. If it was a flight emulator then you'd actually be transported to the destination.

Android Studio

<https://developer.android.com/studio/projects/create-project>

Let's create our first Android app and make sure our environment is all set up.

From the Welcome screen chose Start a new Android Studio Project (File | New | New Project)

Select a Project Template: In the Phone and Tablet tab pick Empty activity.

Name: HelloAndroid

Package name: the fully qualified name for the project

Save location: the directory for your project (make sure there is a directory with the name of the project after the location)

Language: Kotlin

Minimum SDK: API 21: Android 5.0 Lollipop (21 is the minimum API for Material Design)

(Help me choose will show you the cumulative distribution of the API levels)

Leave legacy libraries unchecked (using these restricts usage of some newer capabilities)

Finish

(Preferences | Editor | Color Scheme | General to change scheme to presentation)

Android Studio Tour

<https://developer.android.com/studio/intro/>

The left most pane is the Project Tool Window which provides a hierarchical overview of the project file structure.

The default setting is the Android view which is what I usually leave it in but there are other choices for viewing your project.

Above that is the navigation bar which provides another way to navigate your project files.

Also notice the tabs on the left which will change what is shown in in the left pane.

The middle pane is the editor window which shows the file you're editing. You'll see different editors here depending on what type of file you're editing. If you have more than one file open you will see tabs right above the editor.

Above that is a toolbar for compiling and running your project.

When you're editing a layout's xml file the right pane will show the attributes of whatever view is selected. Above the right pane are controls for the layout view mode.

The bottom window shows the status. There are different tabs at the bottom to choose what you want to see and you can also hide it.

Android Virtual Device (AVD)

<https://developer.android.com/studio/run/managing-avds>

In order to run our app in an emulator we need to create an Android Virtual Device (AVD). An AVD is a configuration that defines a hardware profile, system image, storage area, and other properties of a specific Android device.

Tools | AVD Manager (or AVD Manager  in the toolbar)

Create a virtual device

Category: Phone

Chose a phone like the Pixel - size: large, density: xxhdpi or the Nexus 4 has the size: normal, density: xhdpi

System Image: R API 30 Android 11 or Q API 29 Android 10 (download the needed system image)

AVD Name: Pixel 3 API 289 (or something else descriptive)

Leave the rest as is and Finish

Now you should see your AVD as a virtual device.

Close the AVD Manager.

By default when you run the emulator it runs as a separate application. You can also run the Android Emulator directly in a window in Android Studio using the Emulator tab on the bottom right.

Setup:

File > Settings > Tools > Emulator (or Android Studio > Preferences > Tools > Emulator on macOS), then select Launch in a tool window and click OK.

If the Emulator window doesn't automatically appear, open it by clicking View > Tool Windows > Emulator.

Project Files

Now let's take a closer look at what was created in our project. By default, Android Studio displays your project files in the Android view. This view does not reflect the actual file hierarchy on disk, it is simplified for ease of use.

In the Android view you see 3 main areas of your app

1. manifests: The manifest file describes the fundamental characteristics of the app and defines each of its components. Every part of your app will be declared in this file. It acts as a blueprint for putting all the pieces of an app together.
 - a. Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory.
 - b. The package name is the unique identifier for your app on devices and the Play store
 - c. The application tag describes various characteristics of your app
 - d. Our application also has an activity element with the name MainActivity
 - i. the action MAIN which tells Android which activity should be run when the app starts
 1. Equivalent to a main method used in other languages
 - ii. The category LAUNCHER tells Android that this activity can be launched from the home screen
2. java: These are your Java or Kotlin files that contains all of the core logic for your app. It is named java for historical reasons but also highlights the fact that Kotlin and Java can work interchangeably in Android apps.
 - a. uses the package name
 - b. Unless you're doing testing, you always want to look in the first one (that doesn't say test)
 - c. MainActivity.kt defines a class for us called MainActivity that extends AppCompatActivity.
 - i. AppCompatActivity is a class in Android that is used for all Android activities. It provides access to many features and also provides backwards compatibility with older Android versions

- ii. The : in Kotlin creates a subclass from a superclass
- d. Ctrl-Q (PC), Ctrl-J (Mac) opens the documentation for wherever your cursor is (bottom right tab).
- 3. res: these are static resource files that include
 - a. Drawable – images for the project
 - b. Layout – the layout files for the user interface in xml
 - i. activity_main.xml defines the one TextView that was created for us and its properties including ConstraintLayout properties.
 - c. Mipmap – includes files such as launcher images (app icons are called launcher icons)
 - d. Values – resource files that include values for strings, styles, colors, and other resources
- 4. Gradle: Gradle is the build tool in Android Studio. Gradle controls the following:
 - a. compiles code, dependencies and resources to executable code
 - i. builds the app package file known as an APK file
 - ii. APK is the executable format for distributing Android apps
 - b. Declaring dependent code such as libraries (dependency management)
 - c. Determine what devices the app is built too run on
 - d. App signing for the Google Play store
 - e. Running automated tests

If you switch to the Project view you'll see the actual file structure of the project. There are a lot of other files, but we're not going to worry about most of them.

Run the App


<https://developer.android.com/studio/run>

In the toolbar bar you should see the AVD you just sent up and a play/run button next to it.

Click the Play button to build and run your app.

You might have to wait a bit for the Emulator to start and launch your app.

You can view details in Android Studio about the build process by clicking View > Tool Windows >

Build (or by clicking Build  in the tool window bar at the bottom). The window displays the tasks that Gradle executes in order to build your app,

The first time an emulator starts it can take a few minutes.

(I have found that chromebooks can't always run the emulator so a physical device will be better)

You can leave the emulator running so it will be faster for subsequent runs.

You might need to unlock your device – just swipe the padlock icon upwards and you should see the sample app.

Most newer Android devices have 3 virtual buttons.

Left button (back arrow) goes back to the last activity

Middle button (home) takes you to the home screen

Right button show a list running apps (like a swipe up on iOS)

The favorites tray is at the bottom with the all apps button in the center.

Layout

<https://developer.android.com/studio/write/layout-editor>

Open the activity_main.xml file and the layout opens in the layout editor.




Above the layout editor is a toolbar with buttons to configure the layout editor.

- Design and blueprint
 - Design shows a preview of your layout
 - Blueprint shows only outlines for each view
- Orientation and create layout variants

- Device type and size
- API version
- App theme
- Language

Below those you'll see a bunch of controls for Autoconnect. For now ensure Autoconnect is on by hovering over the magnet looking icon.

To the right and above the toolbar are controls for the view mode

- Design  shows the layout preview
- Code  shows the layout XML
- Split  shows both

At the bottom right of the design editor are zoom and pan controls.

To the right of the design editor is an attributes pane which will show all the attributes for the selected view.

At the top right of that is a tab for layout validation which provides a preview of your layout on different devices. There is also a Gradle tab.

At the bottom right is a tab for the emulator if you want to run it in AS and also a tab for documentation.

To the left of the design editor is the palette which contains all the views and layouts available for use.

Below it is the component tree which shows the hierarchy of components in your layout.

A text view has already been added for us. Click on it in the Component Tree and open up the Attributes pane on the right. Here you can see some of the attributes for the TextView. You can see more by opening different sections including all attributes at the bottom.

Look at the XML for TextView by going into Code mode.

You can see all the same properties including the constraints that were automatically added.

String Resources

Notice the textview has the text property with the value "Hello World" (`android:text="Hello World!"`)

It's not good practice to hard code string values in your xml file, instead you should use string resources. strings.xml in the values folder stores all the string resources.

Find the text property and click the bar next to it.

Click the + in the top left to add a string value.

(In the XML select the string and Option/Alt Enter will bring up a menu to extract string resource)

Resource name: text_message

Resource value: Hello Android

Source set: main

File name strings.xml

Make sure the values directory is checked.

Now the text property should be set to `@string/text_message`

The '@' sign means it's defined as a resource.

Design mode will show that our textview has the new string value.

You can click anywhere on `@string/text_message` and click cmd-B (mac) (ctrl B on a pc) to automatically open the strings.xml file where the values for the string resources are stored.

Open strings.xml and you'll see the resource you just created.

```
<string name="text_message">Hello Android </string>
```

There's also a string for the name of the app.

Back in the layout file let's also make the text larger by either searching in the Properties pane for `textSize` or adding in the xml

`android:textSize="36sp"`

We will always use the "sp" unit for font sizes as it allows the font size to scale for the user based on their settings. (scale-independent pixel)

sp has scalable ratio: $sp = px * ratio * scale$. Where ratio never changes, but scale is user configurable.

This will allow the font size to scale such as for people who set larger font sizes in their settings.

If this is a value you're going to be using throughout your layout you should create an entry in the `dimens.xml` file with a name so you can easily reuse it.

Now let's change the background color through the xml.

Click on `ConstraintLayout` in the component tree or go into the XML.

For the background property you can click on the bar to the right to see all the images and colors already defined in the app or you can use the picker to choose a new color.

`android:background="#ff6232"`

Notice that a small colored square appears on the left.

In the design view you'll see the result.

Click on Run app to run your app in the emulator again (arrow/stop icon). Or stop and then run.

Code completion

The editor in Android Studio has code completion just like in Xcode. As you type you will get selections. To accept the top most suggestion, press the Enter or Tab key on the keyboard. To select a different suggestion, use the arrow keys to move up and down the list, once again using the Enter or Tab key to select the highlighted item.

In the XML file because everything starts with "android" it gets old typing that. You can just start typing what comes after the colon and the code suggestions will still come up. So instead of typing

`android:background` just start typing **`background`** and you'll see **`android:background`** as a choice.

To see other suggestions, click on a word and then use `cntrl-space` and the editor will show you a list of alternative suggestions.

Can you guess how you would change the font color for the textview?

For the TextView add **`android:textColor="#FFFF0D"`**

Note: To open a project in Android Studio there is no one file you can click on to open the project (similar to the `xcodeproj` file). To open a project go into Android Studio and open an existing project from there. If you're opening a project that you did not create, such as my samples, chose import project instead to avoid configuration errors.