**CHAPTER 3: DESCRIPTION OF TOOL**

3.1 Android Studio

Overview:

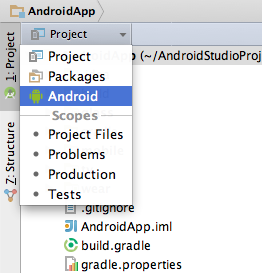
Android Studio is the official IDE for Android application development, based on [IntelliJ IDEA](https://www.jetbrains.com/idea/). On top of the capabilities you expect from IntelliJ, Android Studio offers:

* Flexible Gradle-based build system
* Build variants and multiple apk file generation
* Code templates to help you build common app features
* Rich layout editor with support for drag and drop theme editing
* lint tools to catch performance, usability, version compatibility, and other problems
* ProGuard and app-signing capabilities
* Built-in support for [Google Cloud Platform](http://developers.google.com/cloud/devtools/android_studio_templates/), making it easy to integrate Google Cloud Messaging and App Engine
* And much more

Android Project View:

By default, Android Studio displays your project files in the *Android* project view. This view shows a flattened version of your project's structure that provides quick access to the key source files of Android projects and helps you work with the [Gradle-based build system](http://androiddoc.qiniudn.com/sdk/installing/studio-build.html).

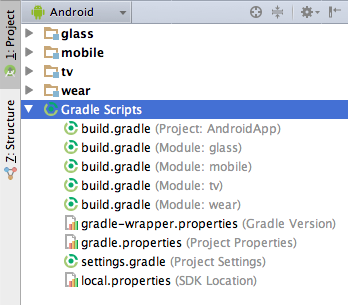
* Shows the most important source directories at the top level of the module hierarchy.
* Groups the build files for all modules in a common folder.
* Groups all the manifest files for each module in a common folder.
* Shows resource files from all Gradle source sets.
* Group’s resource files for different locales, orientations, and screen types in a single group per resource type.



**Figure 3.1. 1.** Show the Android project view.

The Android project view shows all the build files at the top level of the project hierarchy under Gradle Scripts. Each project module appears as a folder at the top level of the project hierarchy and contains these four elements at the top level:

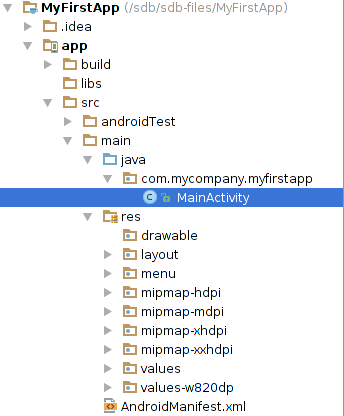
* java/ - Source files for the module.
* manifests/ - Manifest files for the module.
* res/ - Resource files for the module.
* Gradle Scripts/ - Gradle build and property files.



**Figure3.1. 2.** Show project build files

Other Android Views:

When you use the Projectview in Android Studio, you should notice that the project structure appears different than you may be used to in Eclipse. Each instance of Android Studio contains a project with one or more application modules. Each application module folder, including src/main/ and src/androidTest/directories, resources, build file and the Android manifest. For the most part, you will need to modify the files under each module's src/main/ directory for source code updates, the gradle.build file for build specification and the files under src/androidTest/ directory for test case creation.



**Figure 3.1.3.** View Android Studio *Project* structure

You can also customize the view of the project files to focus on specific aspects of your app development:

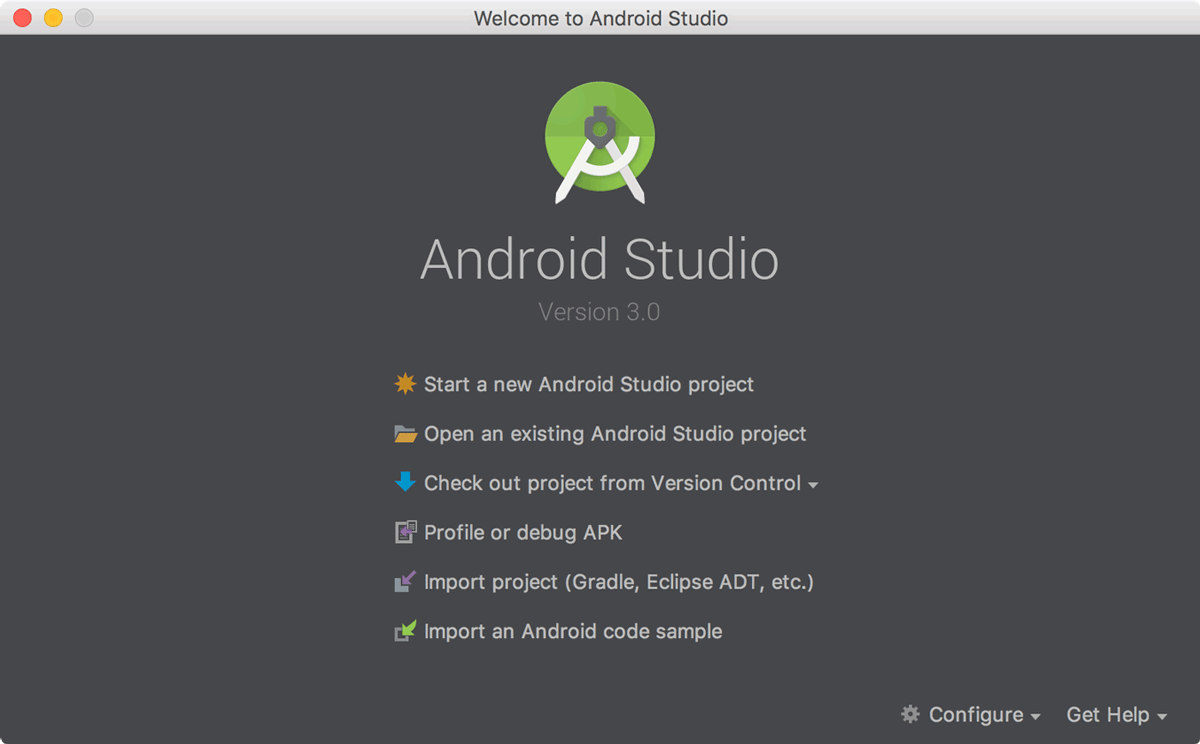
* Packages
* Project Files
* Scratches etc.

3.1.1 Android Applications:

Steps to create new android application:

First, be sure you have installed the latest version of Android Studio. [Download Android Studio here](https://developer.android.com/studio/).

1. In the Welcome to Android Studio window, click Start a new Android Studio project.



**Figure 3.1.1.1.** Android Studio Home page

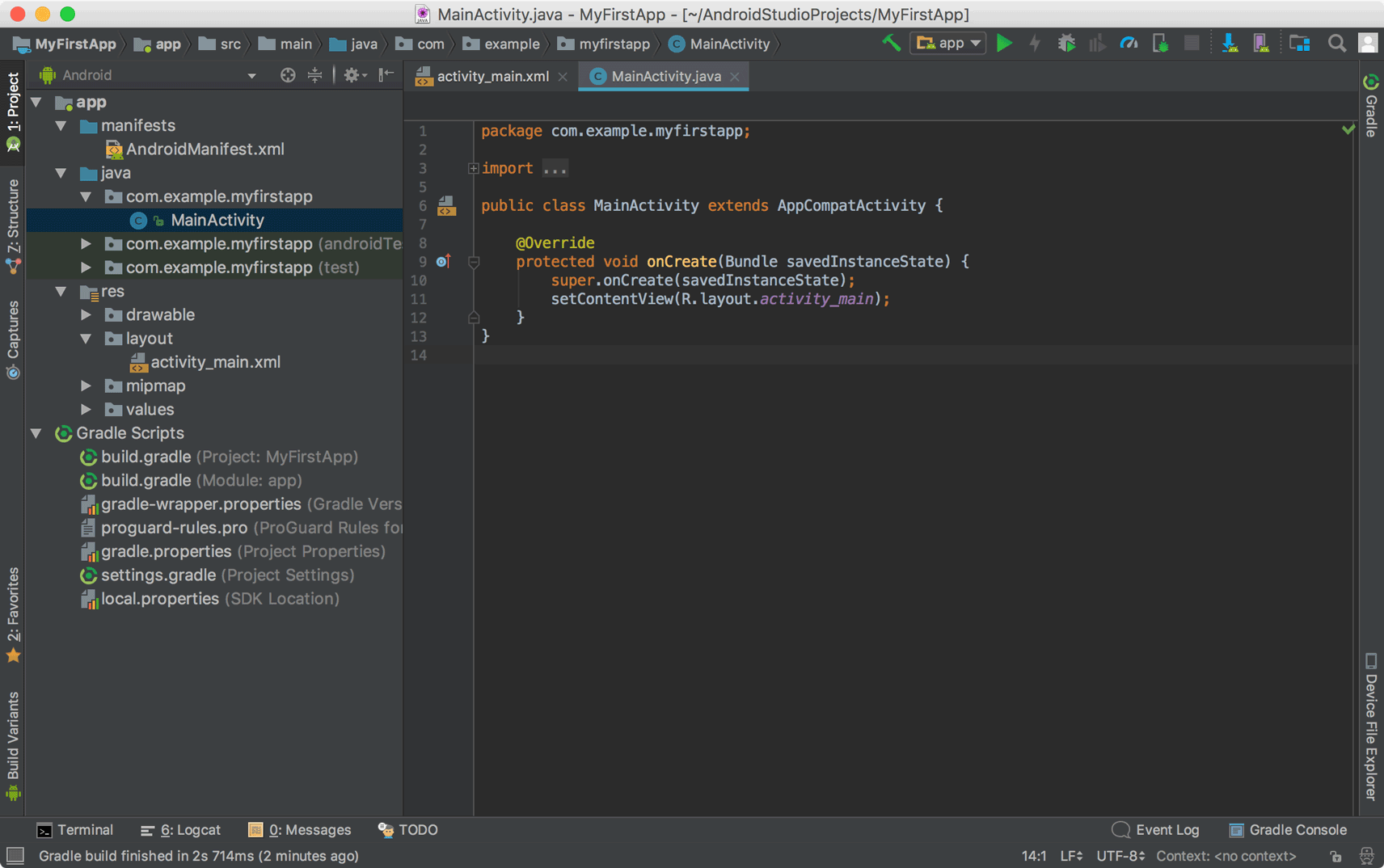
1. Or if you have a project opened, select File > New Project
   1. Application Name: "My First App"
   2. Company Domain: "example.com"

You might want to change the project location. Also, if you want to write a Kotlin app, check the Include Kotlin support checkbox. Leave the other options as they are.

1. Click Next.
2. In the Target Android Devices screen, keep the default values and click Next.
3. In the Add an Activity to Mobile screen, select Empty Activity and click Next.
4. In the Configure Activity screen, keep the default values and click Finish.

After some processing, Android Studio opens the IDE.

Now take a moment to review the most important files.



**Figure 3.1.1.2.** Main Activity

First, be sure the Project window is open (select View > Tool Windows > Project) and the Android view is selected from the drop-down list at the top of that window. You can then see the following files:

app > java > com.example.myfirstapp > MainActivity

This is the main activity (the entry point for your app). When you build and run the app, the system launches an instance of this [Activity](https://developer.android.com/reference/android/app/Activity.html) and loads its layout.

App > res > layout > activity\_main.xml

This XML file defines the layout for the activity's UI. It contains a [Text View](https://developer.android.com/reference/android/widget/TextView.html) element with the text "Hello world!”

app > manifests > AndroidManifest.xml

The [manifest file](https://developer.android.com/guide/topics/manifest/manifest-intro.html) describes the fundamental characteristics of the app and defines each of its components.

Gradle Scripts > build.gradle

You'll see two files with this name: one for the project and one for the "app" module. Each module has its own build.gradle file, but this project currently has just one module. You'll mostly work with the module's build.gradle file to configure how the Gradle tools compile and build your app. For more information about this file, see [Configure Your Build](https://developer.android.com/studio/build/index.html).

Android apps are built as a combination of components that can be invoked individually. For example, an activity is a type of app component that provides a user interface.

The "main" activity is what starts when the user taps your app icon, but you can take the user straight into a different activity from other places, such as from a notification or even from a different app.

Other components such as broadcast receivers and services also allow your app to perform background tasks without a user interface.

After you build your first app, learn more about the other components at [App Fundamentals](https://developer.android.com/guide/components/fundamentals.html).

Apps adapt to different devices

Android allows you to provide different resources for different devices. For example, you can create different layouts for different screen sizes. Then the system determines which layout to use based on the current device's screen size.

If any of your app's features need specific hardware, such as a camera, you can query whether the device has that hardware at runtime and then disable the corresponding features if not. You can also set some features as required so Google Play won't allow installation on devices without them.

After you build your first app, learn more about device configurations at [Device Compatibility](https://developer.android.com/guide/practices/compatibility.html).

3.1.2 Android Programming Language

**JAVA:**

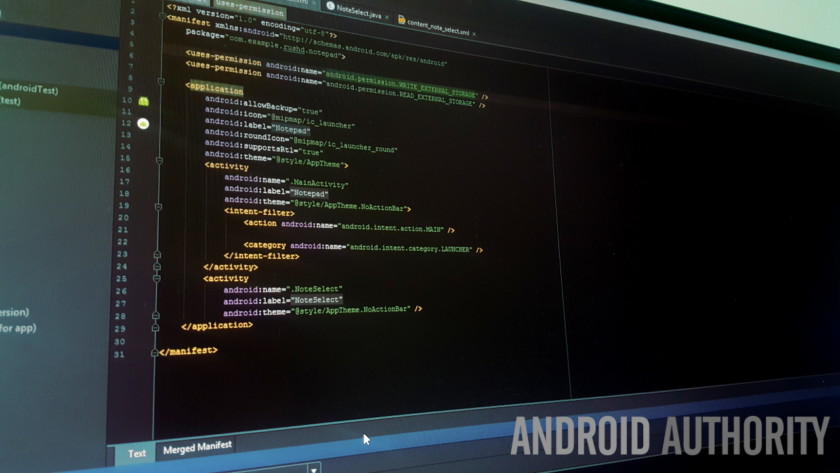
Android is an open source and Linux-based operating system for mobile devices such as smartphones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies. This tutorial will teach you basic Android programming and will also take you through some advance concepts related to Android application development.

The official language for Android development is Java. Large parts of Android are written in Java and its APIs are designed to be called primarily from Java. It is possible to develop C and C++ app using the Android Native Development Kit (NDK), however it isn’t something that Google promotes. According to Google, “the NDK will not benefit most apps. As a developer, you need to balance its benefits against its drawbacks. Notably, using native code on Android generally does not result in a noticeable performance improvement, but it always increases your app complexity.”

Java is a programming language first released by Sun Microsystems back in 1995. It can be found on many different types of devices from smartphones, to mainframe computers. You can use it on your desktop PC and even on the Raspberry Pi. Java doesn’t compile to native processor code but rather it relies on a “virtual machine” which understands an intermediate format called Java byte code. Each platform that runs Java needs a virtual machine (VM) implementation. On Android the original VM is called Dalvik. Google has also started previewing its next generation VM called ART. The job of these virtual machines is to interpret the byte code, which is really just a set of instructions similar to the machine code found in CPUs, and executes the program on the processor. The VMs use a variety of technologies including just-in-time compilation (JIT) and ahead-of-time compilation (AOT) to speed up the processes.

When it comes time to develop Android apps, the first and most popular option is Java. [Java](https://www.androidauthority.com/java-tutorial-beginners-2-582147) is the **official** language of Android development, meaning it is the one that has the most support from Google and the one that most apps on the Play Store are built with.

The number one way to develop Android apps is to go ahead and download [Android Studio](https://www.androidauthority.com/android-studio-tutorial-beginners-637572/). This is a piece of software called an IDE, or Integrated Development Environment. It will come packaged with the [Android SDK](https://www.androidauthority.com/android-sdk-tutorial-beginners-634376/) (a set of tools to facilitate Android development specifically) and basically this will give you everything you need in one place to get up and running.



**Figure 3.1.2.1: Java language**

The [official tutorials and documentation](https://developer.android.com/index.html) from Google will reference this method and you’ll find the largest number of libraries (free code to enhance your own apps) and tutorials that focus on this method.

Java itself was released by Sun Microsystems back in 1995 and is used for a wide range of programming applications. Java code is run by a “virtual machine,” which runs on Android devices and interprets the code.

Unfortunately, Java is also a little complicated and it’s not a great “first language.” This is what will provide the biggest barrier for many people who want to get started with Android development, in fact. Android is an object oriented programming language with confusing topics like constructors, null pointer exceptions, checked exceptions and more. It’s not terribly readable and you’ll use a lot of “boiler plate” code doing simple things. Add in the Java SDK and things get more complicated still – a first time coder can struggle to know what Java is and what Android is! Development using this route also requires a basic understanding of concepts like Gradle, like the Android Manifest and the mark-up language XML.

**KOTLIN:**

Kotlin recently burst onto the scene as the “other” official language for Android development. Some speculation suggests that this is likely to raise the language’s profile and that it could possibly become the next Swift.

Like Java, Kotlin runs on the Java Virtual Machine. It’s also completely interoperable with Java and doesn’t cause any slow down or increase in file sizes. The difference is that Kotlin requires less “boiler plate” code, meaning that it is a more streamlined and easy-to-read system. It also does away with errors like null point exceptions and even excuses you from ending every line with semi colons. In short, it’s great if you’re just learning to develop Android apps for the first time.

So Kotlin is definitely an easier starting point for beginners, and the fact that you can still use Android Studio is a big plus. It’s still not quite as simple to pick up as say C# with Unity though, and the community support is in its relative infancy. In fact, you currently need to download a beta version of Android Studio in order to get the out-the-box support.

The other languages you might consider learning for Android development include:

C/C++ — Android Studio also supports C++ with the use of the Java NDK. This allows for native coding applications, which can be handy for things like games. C++ is more complicated though.

C# — C# is a slightly more beginner-friendly alternative to C or C++ that obfuscates more code. It’s supported by some very handy tools like Unity and Xamarin which are great for game development and for cross-platform development.

BASIC – A bonus option is to learn BASIC and try the B4A IDE from Anywhere Software. This is an easy but powerful tool, though definitely much more niche!

Corona/LUA – Another cross-platform tool build on LUA. It massively simplifies the app-building process and allows you to call native libraries.

3.1.3 Database:

Saving data to a database is ideal for repeating or structured data, such as contact information. This page assumes that you are familiar with SQL databases in general and helps you get started with SQLite databases on Android. The APIs you'll need to use a database on Android are available in the [android.database.sqlite](https://developer.android.com/reference/android/database/sqlite/package-summary.html) package.

One of the main principles of SQL databases is the schema: a formal declaration of how the database is organized. The schema is reflected in the SQL statements that you use to create your database. You may find it helpful to create a companion class, known as a *contract* class, which explicitly specifies the layout of your schema in a systematic and self-documenting way.

A contract class is a container for constants that define names for URIs, tables, and columns. The contract class allows you to use the same constants across all the other classes in the same package. This lets you change a column name in one place and have it propagate throughout your code.

A good way to organize a contract class is to put definitions that are global to your whole database in the root level of the class. Then create an inner class for each table. Each inner class enumerates the corresponding table's columns.

The [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html) class contains a useful set of APIs for managing your database. When you use this class to obtain references to your database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup.

To use [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html), create a subclass that overrides the [onCreate()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#onCreate(android.database.sqlite.SQLiteDatabase)) and [onUpgrade()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#onUpgrade(android.database.sqlite.SQLiteDatabase,%20int,%20int)) callback methods. You may also want to implement the[onDowngrade()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#onDowngrade(android.database.sqlite.SQLiteDatabase,%20int,%20int)) or [onOpen()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#onOpen(android.database.sqlite.SQLiteDatabase)) methods, but they are not required.

public class FeedReaderDbHelper extends SQLiteOpenHelper {  
    // If you change the database schema, you must increment the database version.  
    public static final int DATABASE\_VERSION = 1;  
    public static final String DATABASE\_NAME = "FeedReader.db";  
  
    public FeedReaderDbHelper(Context context) {  
        super(context, DATABASE\_NAME, null, DATABASE\_VERSION);  
    }  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL(SQL\_CREATE\_ENTRIES);  
    }  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        // This database is only a cache for online data, so its upgrade policy is  
        // to simply to discard the data and start over  
        db.execSQL(SQL\_DELETE\_ENTRIES);  
        onCreate(db);  
    }  
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        onUpgrade(db, oldVersion, newVersion);  
    }  
}

To access your database, instantiate your subclass of [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html):

FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());