**SLIDE 7 – activity\_main.xml**

<https://developer.android.com/guide/topics/ui/declaring-layout>

* A layout defines a structure for a user interface in your app such as an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something a user can see and interact with, a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects. View object = widget, ViewGroup = layout.
* Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses. You can also use Android Studio’s layout editor to build your layout using a drag and drop interface.
* Declaring your UI in XML allows you to separate the presentation of your app from the code that controls its behavior. Using XML files also makes it easy to provide different layouts for different screen sizes and orientations.
* Using Android’s XML vocabulary, can quickly design UI layouts and screen elements in the same way you create web pages using HTML – a series of nested elements. Each layout file must contain exactly one root element, which must be a View or a ViewGroup object. You can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.
* When you compile your app, each XML layout file is compiled into a View resource. Load the layout resource in onCreate(), by passing setContentView() a reference to your layout resource in the form of R.layout.layout\_file\_name.

**SLIDE 8 – AndroidManifest.xml**

The primary task of the manifest is to inform the system about the app's components.

Every app project must have an AndroidManifest.xml at the root of the project source set. The manifest describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

Android build tools use the **package** attribute for two things:

* Applies this name as the namespace for your app’s generated R.java class (used to access resources), in this example com.russ.helloworld.R is created
* uses this name to resolve any relative class names declared in the manifest file. Example: an activity declared as <activity android:name=”.MainActivity”> is resolved to com.example.myapp.MainActivity

Before the Android system can start an app component, the system must know that the component exists by reading the app's *manifest file*, AndroidManifest.xml. Your app must declare all its components in this file, which must be at the root of the app project directory.

The manifest does a number of things in addition to declaring the app's components, such as the following:

* Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
* Declares the minimum [API Level](https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels) required by the app, based on which APIs the app uses.
* Declares hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.
* Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the [Google Maps library](http://code.google.com/android/add-ons/google-apis/maps-overview.html).

**SLIDE 9 – build.gradle**

Gradle is a build system. Build systems are software tools designed to automate the process of program compilation.

Build system takes all source files (.java, .xml, .kt) and applies the appropriate tool (ie converts .java to dex file, and then groups them all into the apk

Google wrote the Android plugin for Gradle because they realized you could write scripts of your own with little or no learning curve and without learning Groovy/any new language.

* Build.gradle is a build script
* Build.gradle defines a project and its tasks
* Build scripts are code (Groovy).

Gradle build files use a domain specific language (DSL) to define custom build logic and interact with Android-specific elements of the Android plugin for Gradle.

Android studio projects consist of 1 or more modules, which are components you can build, test, and debug independently. Each module has its own build file, so every Android studio project contains 2 types of Gradle build files:

* Top-level build file – configuration options common to all the modules that make up your project
* Module-level build file – each module has its own Gradle build file that contains module-specific build settings. Most time will be spent here, rather than in top-level build file.

Other Gradle files:

* Gradle-wrapper.properties – this file allows other people to build your code, even if they don’t have gradle installed on their machine. Checks whether correct version of Gradle is installed, and downloads the correct version if necessary.
* Settings.gradle – this file references all the modules that make up your project.
* Gradle.properties (Project Properties) – this file contains configuration information for your entire project. Empty by default, but you can apply a wide range of properties to your project by adding them to this file.
* Local.properties (SDK location) – tells the Android Gradle plugin where it can find your Android SDK installation. Contains information local to the Android SDK installation, so don’t keep it under source control.