Android Application Development, COMP 10073

Mohawk College, Winter 2021

MainActivity

Using the instructions from the first lecture, create a project using the Empty Activity template and set it up for API 23: Android 6.0 (Marshmallow). Your Android Virtual Device (AVD) should also be configured as an Android 6.0 device. The Empty Activity template provides a collection of XML resources, configuration files, a MainActivity.java file, and some gradle build files.

Recipe Approach

We will investigate Android application development from a recipe perspective. Much of what we will do this term is learn about the Android Application framework through individual examples. Each example contributes several basic building blocks used in Android Applications.

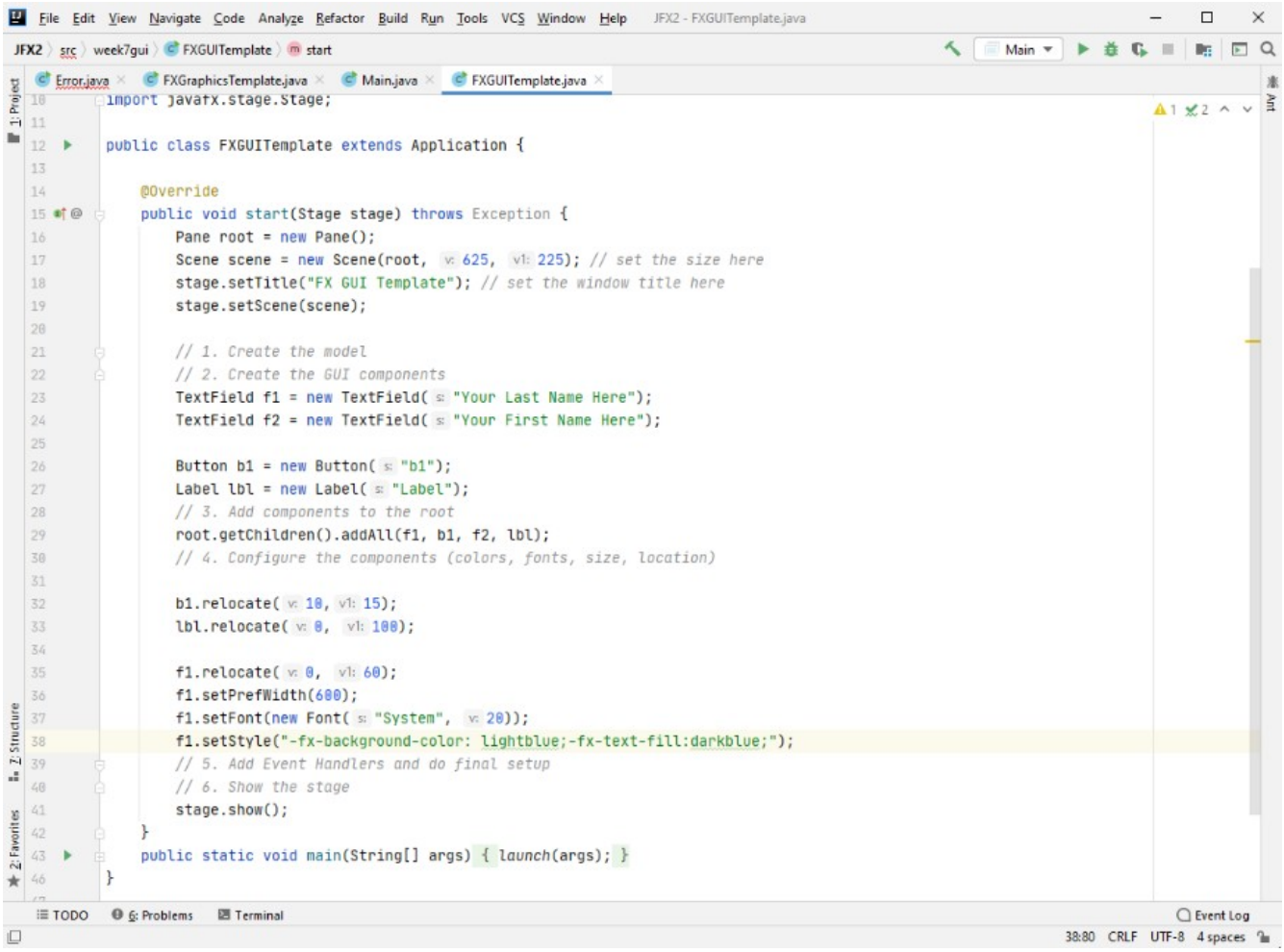
Android applications do not include a main method. Android apps are GUI driven, and are more like JavaFX applications. New events that occur on the device, like starting an application, rotating the phone, receiving an SMS message, or a telephone call, trigger certain methods and will often launch an activity.

In the Hello World example, the **onCreate(...)** method is called when the application starts. It takes as its argument a “Bundle” which contains information defining the applications previous state. It then calls **setContentView(...)**.

The method setContentView is inherited from **AppCompatActivity**, it displays the contents of **R.layout.activity_main** which includes the Hello World message.

Compare onCreate() with JavaFX start()

All GUI environments are platform specific, and while there are similar concepts across GUIs, recall that the JavaFX GUI initialization did something strange with the main method, it called **launch(...)**, which was inherited from the Application class.

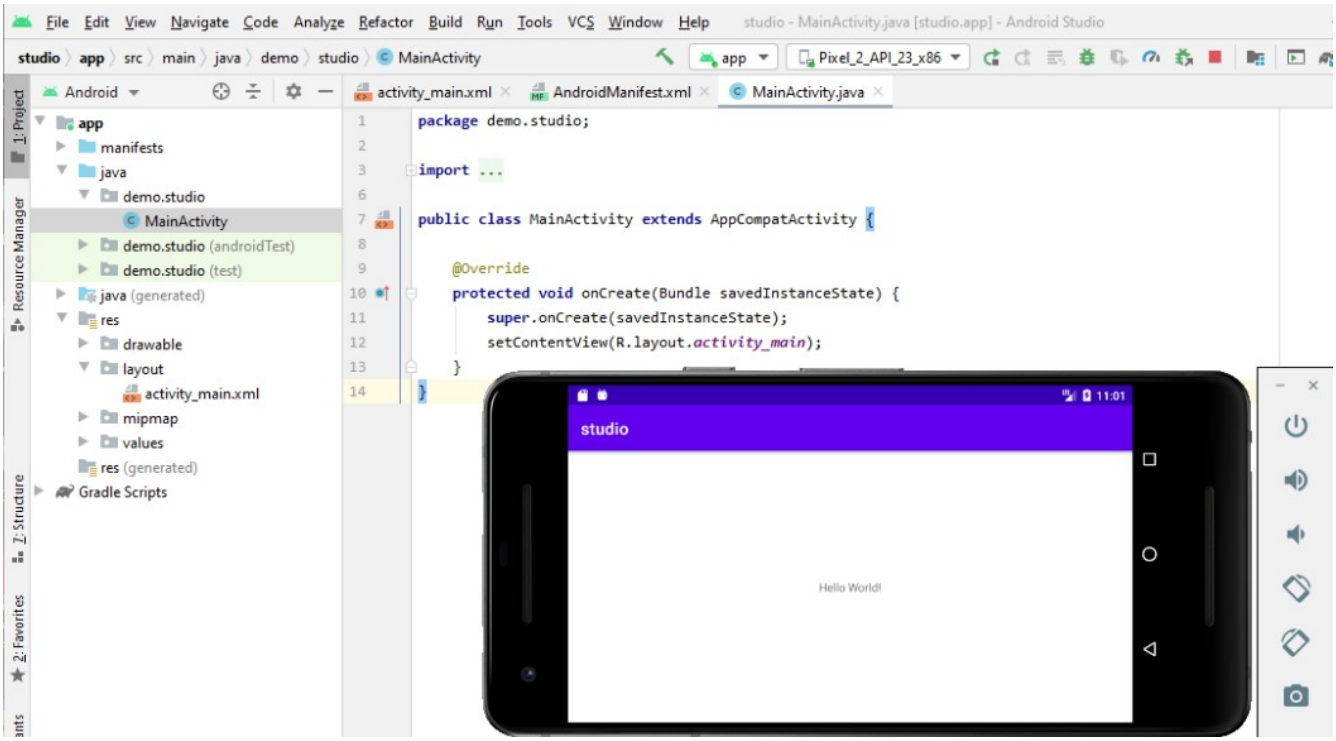


Somehow, JavaFX applications call the **start(...)** method which you provide by overriding an existing template definition. The launch() method effectively gets the JavaFX system up and running and it expects you to override the start(...) method with your own program.

In the same way that you created a graphical context and drew images and widgets on the JavaFX “Pane”, and defined callback methods for those widgets, we will do something similar with Android.

The Java course follows a programmatic implementation of widgets, where you create objects and place them in fixed positions on the interface. For the Android course we will use a special editor that allows a more flexible approach.

Android applications must support a wide array of device formats. It is normal to be able to rotate a phone or tablet and have the application recognize the change in orientation and adjust itself appropriately.



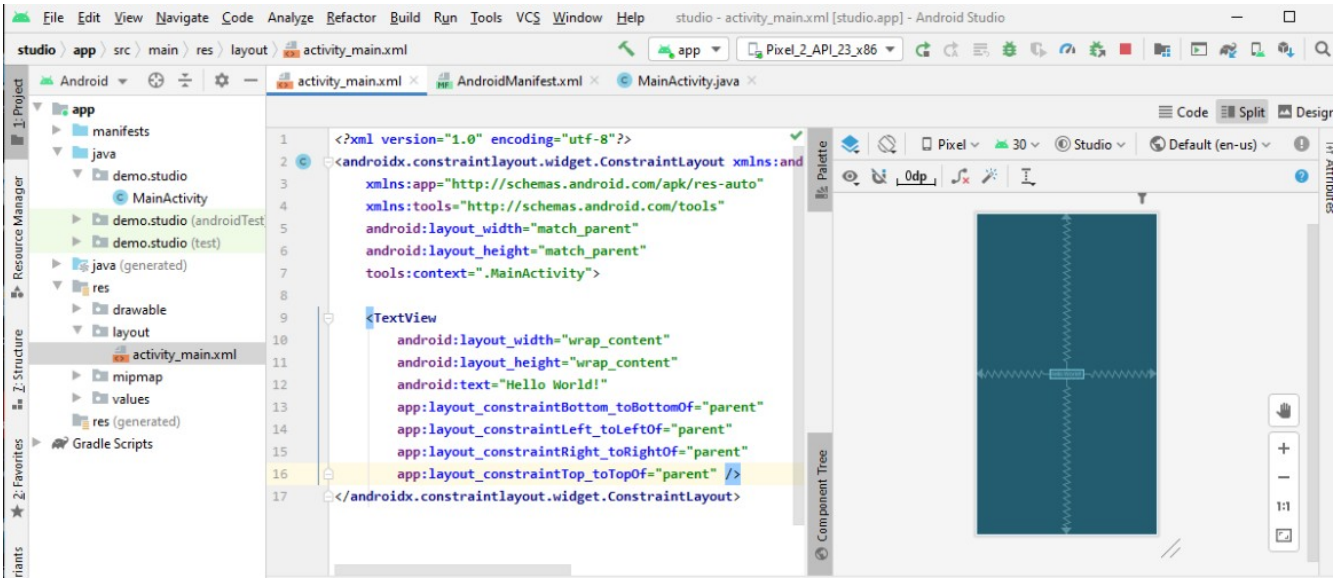
An Activity is defined in Android as a screenfull, it is the basic unit of a GUI program. The **MainActivity** is the common starting point for an Android application. In this case the onCreate() method is called when the application is started and also when the phone is rotated.

Rotating the phone requires the text to be redrawn, and repositioned. If we had to hard code all of these numbers building a portable Android application that worked in a variety of environments would be a huge amount of work.

Instead we call **setContentView** and pass in a reference to **R.layout.activity_main**. Every Android application provides a collection of resources including layouts. Under the “res” directory, you will find a layout directory which includes an XML file which defines the contents for “activity_main”. There is a one-to-one correspondence between this directory hierarchy and the symbolic name of the resource.

Android Application Layout

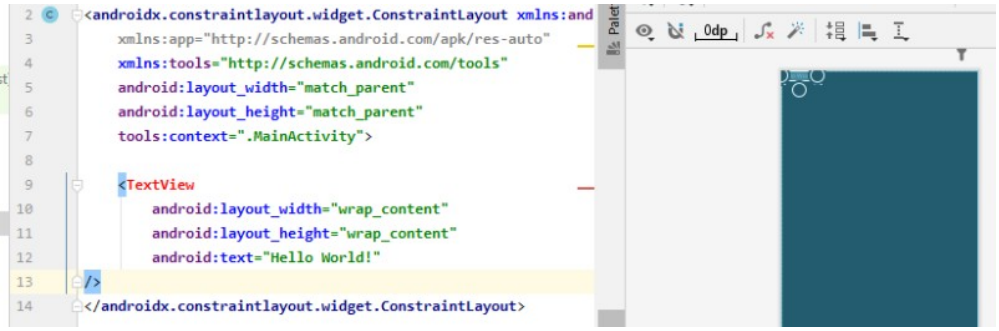
Widget layouts in Android are defined by XML files. The Android XML editor will show you both a rendered version of your application as well as the XML code that defines the layout.



The Hello World application uses a “constraintlayout” template for its MainActivity. This allows the position of the text to be defined in a relative fashion. The activity uses a width and height that matches the parent. This allows the activity to work on any device of any size and orientation.

The initial text for the text widget is in this case hard-coded into the XML. You can adjust it in the XML editor and the design image will adjust accordingly.

Likewise the position of the text in within the activity is defined through connections between widgets. The top of the text widget is connected to the bottom of the parent, and so on. If you remove these constraints the widget will use its default position of the upper right corner.



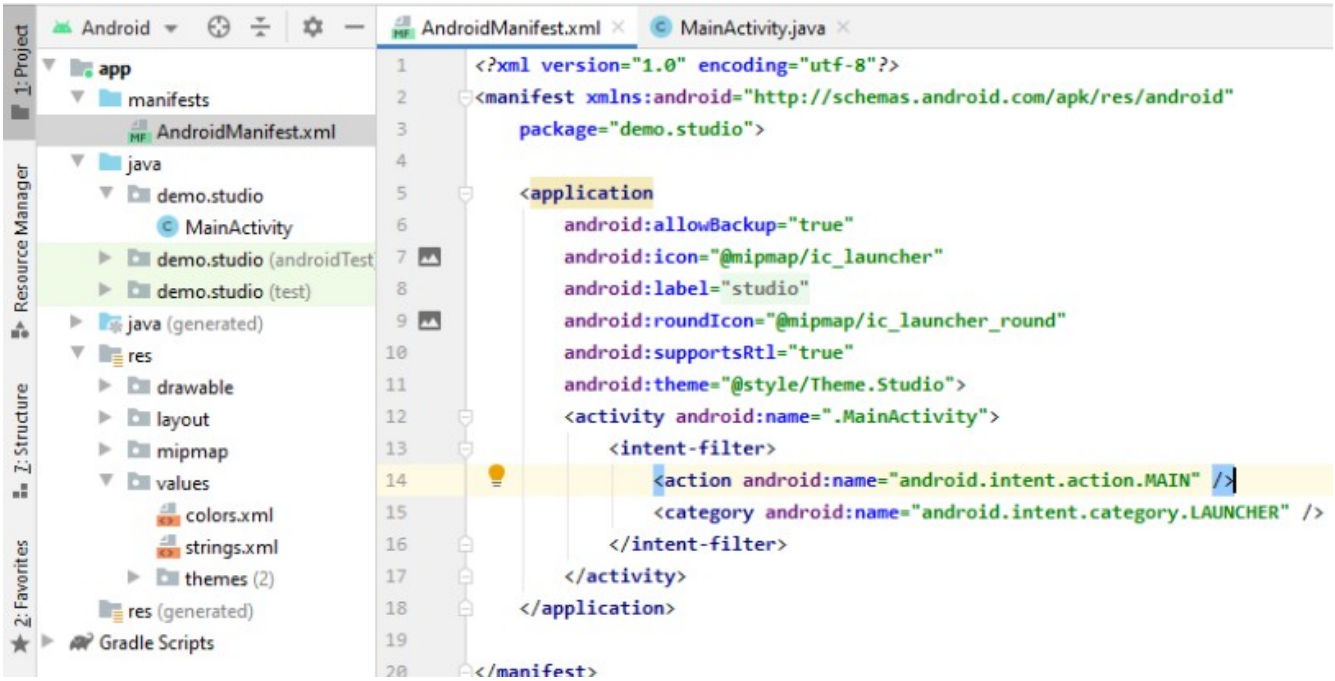
Manifest File

https://developer.android.com/guide/topics/manifest/manifest-intro

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

Among many other things, the manifest file is required to declare the following:

- The app's package name, which usually matches your code's namespace. The Android build tools use this to determine the location of code entities when building your project.
- The components of the app, which include all activities, services, broadcast receivers, and content providers. Each component must define basic properties such as the name of its Kotlin or Java class.
- The permissions that the app needs in order to access protected parts of the system or other apps.
- The hardware and software features the app requires, which affects which devices can install the app from Google Play.



Manifest: Application Properties

https://developer.android.com/guide/topics/manifest/application-element

The manifest file for the demonstration includes the application element which defines a collection of properties:

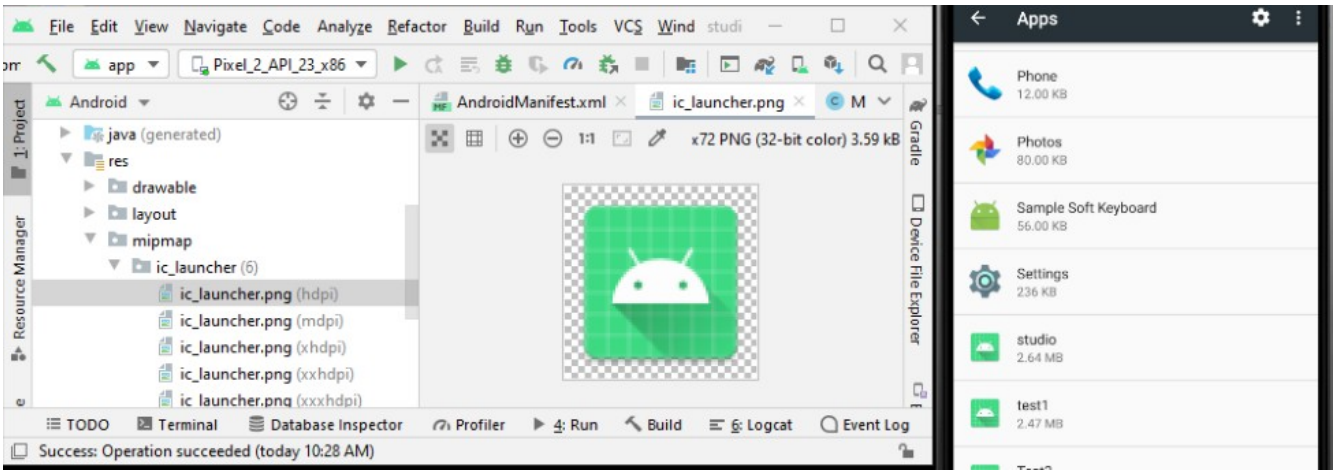
android:allowBackup

Whether to allow the application to participate in the backup and restore infrastructure. If this attribute is set to false, no backup or restore of the application will ever be performed.

android:icon

An icon for the application as whole, and the default icon for each of the application's components.

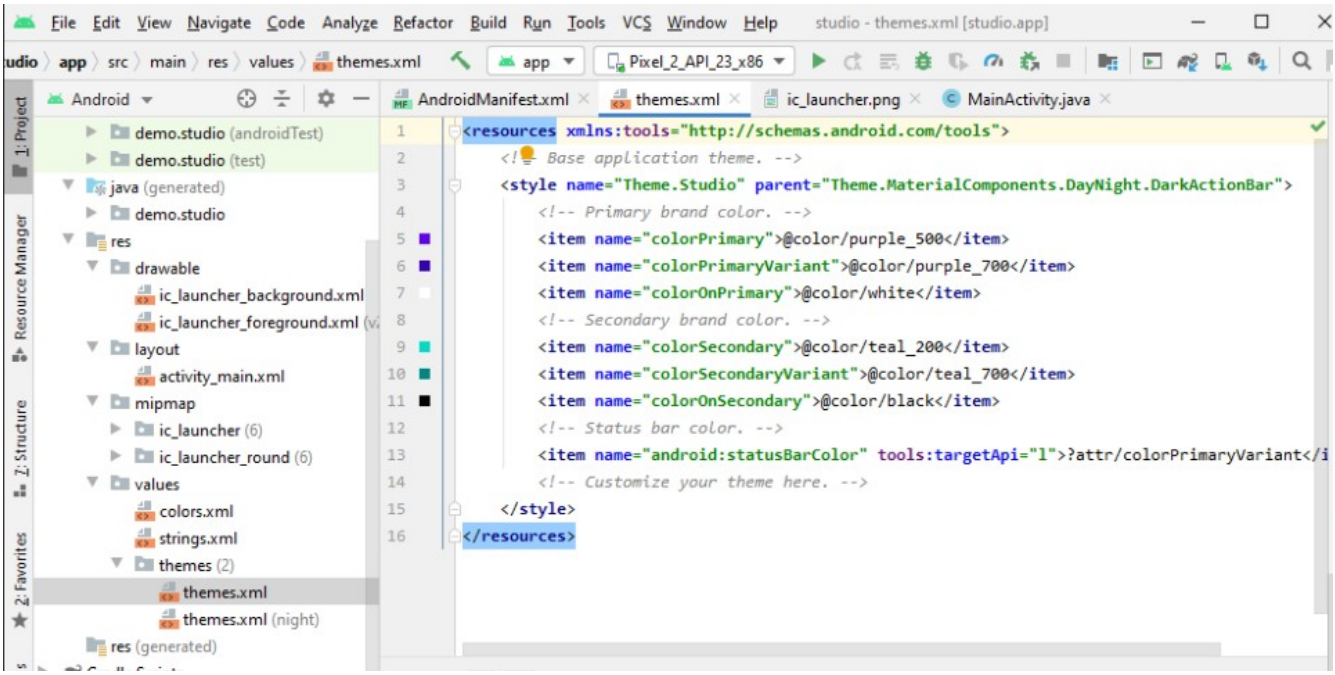
Note in this example that the icon is defined as “@mipmap/ic_launcher”, the “@” syntax provides an XML reference to an internal resource, similar to the “R.*” syntax that we mentioned previously. In this case if we look under “mipmap/ic_launcher” we can find the icon used to represent our test application:



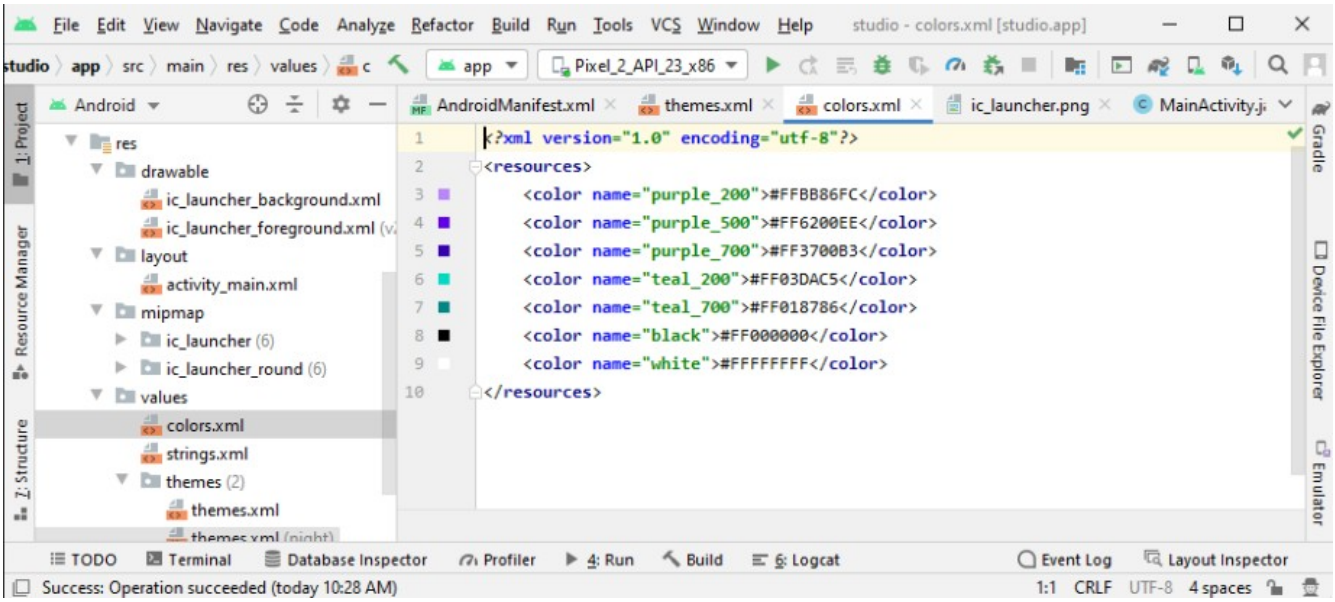
android:theme

A reference to a style resource defining a default theme for all activities in the application. A style is a collection of attributes that specify the appearance for a single View. A theme is a collection of attributes that's applied to an entire app, activity, or view hierarchy. When you apply a theme, every view in the app or activity applies each of the theme's attributes that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

Themes are declared in a resource file in **res/values/themes.xml**.



In this example “Theme.Studio” defines a collection of colors. The colors themselves are illustrated in the margin beside the line numbers which is a neat feature of Android Studio. The color definitions themselves can be found in the colors.xml file.



Manifest File: Activity Element

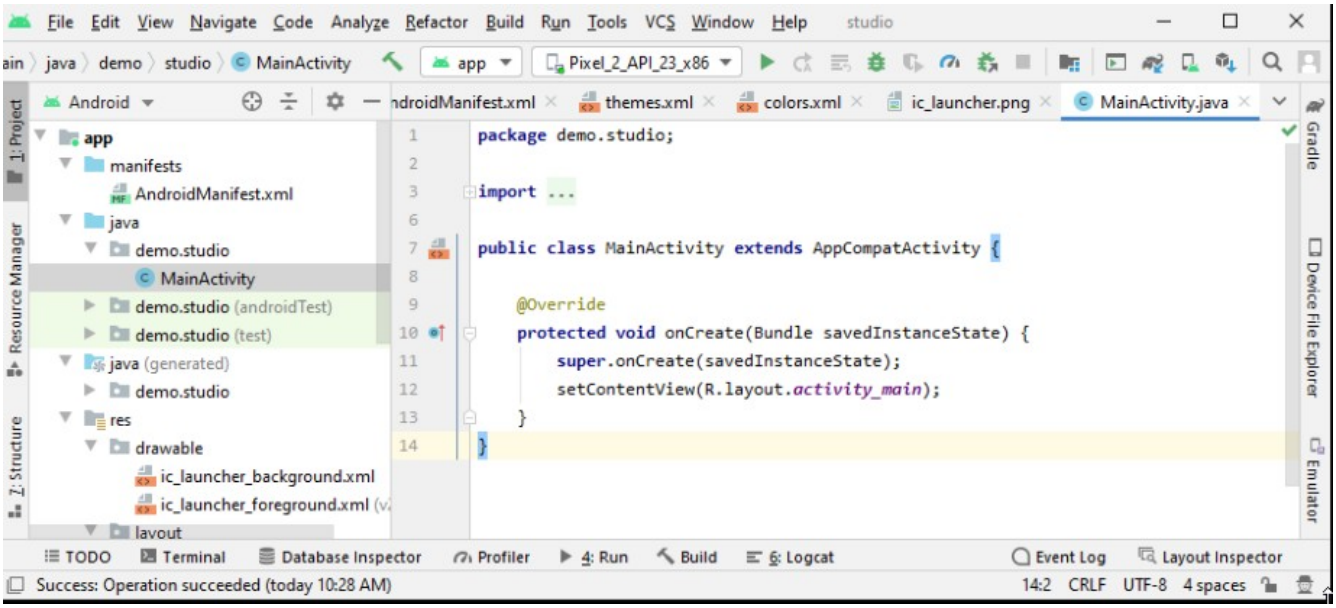
https://developer.android.com/guide/topics/manifest/activity-element

```
11          android:theme="@style/Theme.Studio">
12          <activity android:name=".MainActivity">
13              <intent-filter>
14                  <action android:name="android.intent.action.MAIN" />
15                  <category android:name="android.intent.category.LAUNCHER" />
16              </intent-filter>
17          </activity>
```

The **Activity** Element declares an activity (an Activity subclass) that implements part of the application's visual user interface. All activities must be represented by <activity> elements in the manifest file. Any that are not declared there will not be seen by the system and will never be run.

android:name=".MainActivity"

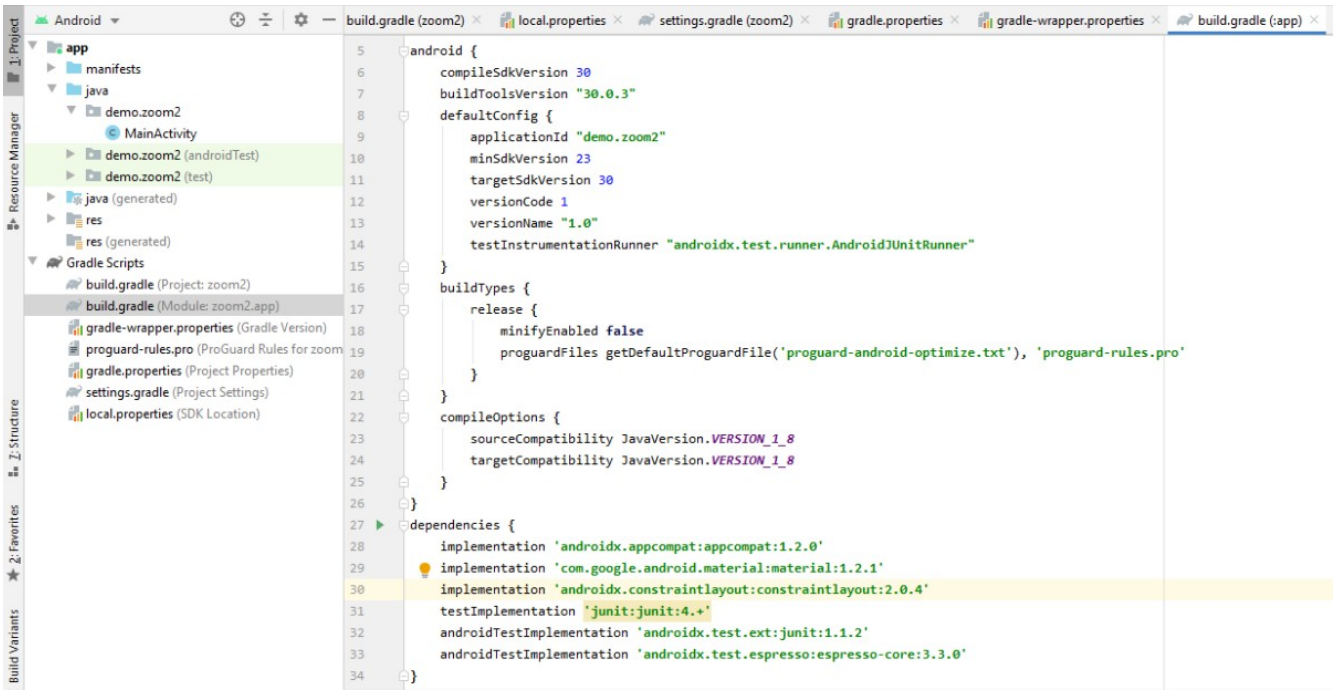
The name attribute defines the class that implements the activity. This takes us to our initial Java file:



Gradle Files

https://developer.android.com/studio/build

Android Studio uses Gradle, an advanced build toolkit, to automate and manage the build process, while allowing you to define flexible custom build configurations. Each build configuration can define its own set of code and resources, while reusing the parts common to all versions of your app.



Gradle and the Android plugin run independent of Android Studio. This means that you can build your Android apps from within Android Studio, use the command line on your machine, or on machines where Android Studio is not installed (such as continuous integration servers).

Usually we can ignore these files. Android studio will automatically manage them for us, although sometimes we will need to tweak dependencies to ensure that certain widgets are working. Build errors that complain about missing components are sometimes the result of missing dependencies.

In the next lecture we will look at working with layouts, in particular the constrained layout is important for us to understand.