Creating a Project

We'll cover the following

- Initialize the application
- Reviewing files
- Adding additional dependencies

Initialize the application

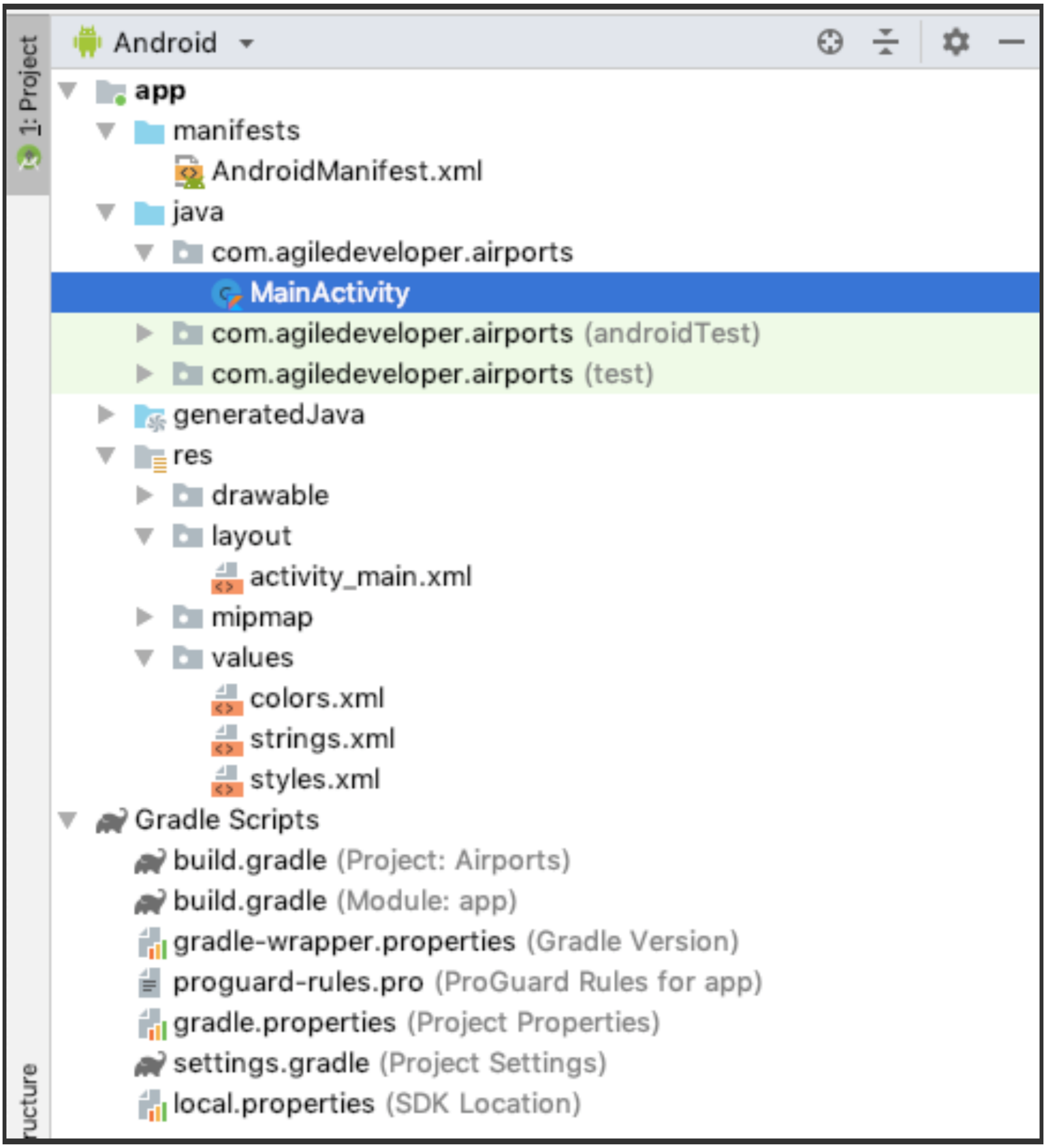
If you want to set up the application environment on your system then follow along with the upcoming instruction.

The Android Studio is a pretty good IDE to create Android applications. Download the latest version of the IDE if you don't have that already on your system. Fire up the IDE to create a new project. If the IDE doesn't open an existing project, click on the link "Start a new Android Studio project." If you already have a project open, then click on the `File | New | New Project...` menu items. Irrespective of the path you take to create a new project, in the next step, in the Choose your Project dialog select Phone and Tablet, and then select Empty Activity. Finally, click the Next button.

Name the project "Airports" and the package name `com.agiledeveloper.airports`. Provide an appropriate location to save your project. Make sure to select Kotlin as the language in the list box under the Language label. For the Minimum API Level, select API 28: Android 9.0 (Pie). Click on Finish to create the project.

Reviewing files

Let's review the interesting parts of the project from within Android Studio.



```
Android
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com.agiledeveloper.airports
        MainActivity
    ▶ com.agiledeveloper.airports (androidTest)
    ▶ com.agiledeveloper.airports (test)
  ▶ generatedJava
  ▼ res
    ▶ drawable
    ▼ layout
        activity_main.xml
    ▶ mipmap
    ▼ values
        colors.xml
        strings.xml
        styles.xml
▼ Gradle Scripts
    build.gradle (Project: Airports)
    build.gradle (Module: app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

In the Project pane on the left, in the Android section, take note of the contents under app.

- The manifests folder contains the file `AndroidManifest.xml`, which we'll edit soon to enable network access from within the Android emulator.
- Even though we're using Kotlin, the IDE is storing the source files under java but within the package name we provided.

- Under layout there is currently one layout file `activity_main.xml` for the empty activity we created—we'll edit this file later.
- Under values are some files that can be used to tailor settings for internationalization. The `strings.xml` file contains the name of the application, Airports, that will be displayed at the top of the application.
- Under Gradle Scripts are two `build.gradle` files. The one annotated with "(Project: Airports)" is the top-level build file, and the one with the words "(Module: app)" is the subproject-level build file. Soon we'll edit the latter to add dependencies needed for the application.

Adding additional dependencies

Before going any further, let's update a few dependencies we'll need as soon as we start coding the app.

First, add dependency to the Klaxon library that will be used to parse the JSON response from the web service. For this, double click the file `build.gradle` (Module: app). Scroll down in the file to the `dependencies` section, and right after the other lines that start with `implementation`, add the following line:

```
implementation "com.beust:klaxon:5.0.2"
```

In addition to using the Klaxon library, our app will also use Kotlin coroutines. To facilitate that, add these two lines right after the line that specifies the Klaxon dependency:

```
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.2.2"
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.2.2"
```

The first is for the Kotlin's coroutines library, and the second is for Android-specific coroutines integration that's needed to support coroutines in Android.

After making these changes, we need to tell the IDE to refresh the Gradle build file and download the dependencies. For this, in the IDE, click the File menu and select Sync Project with Gradle Files. Wait a moment for the IDE to refresh the project, and download the dependencies we've introduced.

One final configuration before we can move on to the coding—we'll be running the app within an Android emulator. By default, the code running within the emulator doesn't have unrestricted access to the network. For the app to be able to get data from the remote web service, we have to modify the file `AndroidManifest.xml`,

which you can find in the Project pane under app/manifest. Open that file, and add the following line right before the `</manifest>` closing tag:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Now that the project is set up and the necessary dependencies and configurations are in place, let's move on to the code in the next lesson.