Unit-2: Android Project Structure and Basics

2

Unit Structure

2.0 Learning Objectives

After studying this unit student should be able to:

- Know the structure of Android Project
- List the various types of Android Project files
- Define Android application modules
- Enumerate the types of modules
- Modify project structure setting
- Understand anatomy of an android application
- Know basic and advanced Android API

2.1 Introduction

The Android build system is organized around a specific directory tree structure for the Android project, similar to the any Java project. The project prepares the actual application that will run on the device or emulator. When you create a new Android project, you get several items in the project's root directory which is discussed in sub sequent sections.

When you create an Android project as discussed in previous unit, you provide the fully-qualified class name of the "main" activity for the application (e.g., edu.baou.HelloWorld).

You will then find that your project's src/ tree already has the namespace directory tree in place, plus a stub Activity subclass representing your main activity (e.g., src/edu/baou/HelloWorld.java). You can modify this file and add others to the src/ tree as per requirement implement your application.

When you compile the project for first time, in the "main" activity's namespace directory, the Android build chain will create R.java. This contains a number of constants tied to the various resources you placed out in the res/ directory tree. You should not modify R.java yourself, letting the Android tools handle it for you. You will see throughout many of the samples where we reference things in R.java (e.g., referring to a layout's identifier via R.layout.main).

2.2 Android Project Structure

An Android project contains everything that defines your Android app. The SDK tools require that your projects follow a specific structure so it can compile and package your application correctly. Android Studio takes care of all this for you.

A module is the first level of control within a project that encapsulates specific types of source code files and resources. There are several types of modules with a project:

Module	Description
Android Application Modules	It contain source code, resource files, and application level settings, such as the module-level build file, resource files, and Android Manifest file.
Test Modules	It contains code to test your application projects and is built into test applications that run on a device.
Library Modules	It contains shareable Android source code and resources that you can reference in Android projects. This is useful when you have common code that you want to reuse.
App Engine Modules	They are App Engine java Servlet Module for backend development, App Engine java Endpoints Module to convert server-side Java code annotations into RESTful backend APIs, and App Engine Backend with Google Cloud Messaging to send push notifications from your server to your Android devices.

Table-4

When you use the Android development tools to create a new project and the module, the essential files and folders will be created for you. As your application grows in complexity, you might require new kinds of resources, directories, and files.

2.3 Android Project Files

Android Studio project files and settings provide project-wide settings that apply across all modules in the project.

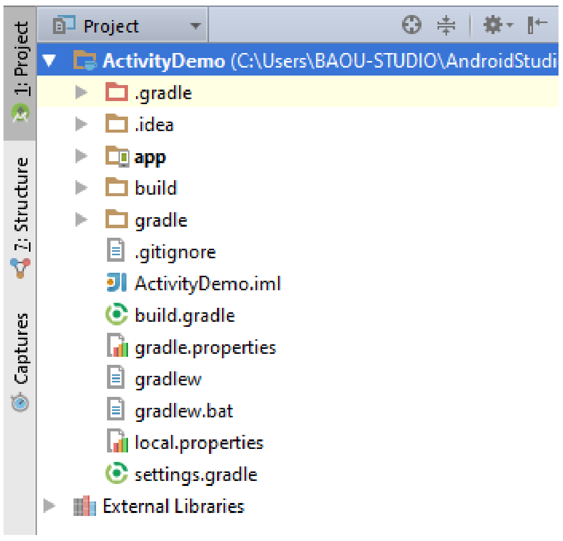

Figure-43

File	Meaning
.idea	Directory for IntelliJ IDEA settings.
App	Application module directories and files.
Build	This directory stores the build output for all project modules.
Gradle	Contains the gradler-wrapper files.
.gitignore	Specifies the untracked files that Git should ignore.
build.gradle	Customizable properties for the build system.
gradle.properties	Project-wide Gradle settings.
gradlew	Gradle startup script for Unix.
gradlew.bat	Gradle startup script for Windows.
local.properties	Customizable computer-specific properties for the build system, such as the path to the SDK installation.
.iml	Module file created by the IntelliJ IDEA to store module information.
settings.gradle	Specifies the sub-projects to build.

Table-5

Check your progress-1

a) __________contains shareable Android source code and resources that you can reference in Android projects.

b) __________ contain source code, resource files, and application level settings, such as the module-level build file, resource files, and Android Manifest file.

c) When you compile the project for first time, in the "main" activity's namespace directory, the Android build chain will create __________

2.4 Android Application Modules

Android Application Modules contain things such as application source code and resource files. Most code and resource files are generated for you by default, while others should be created if required. The following directories and files comprise an Android application module:

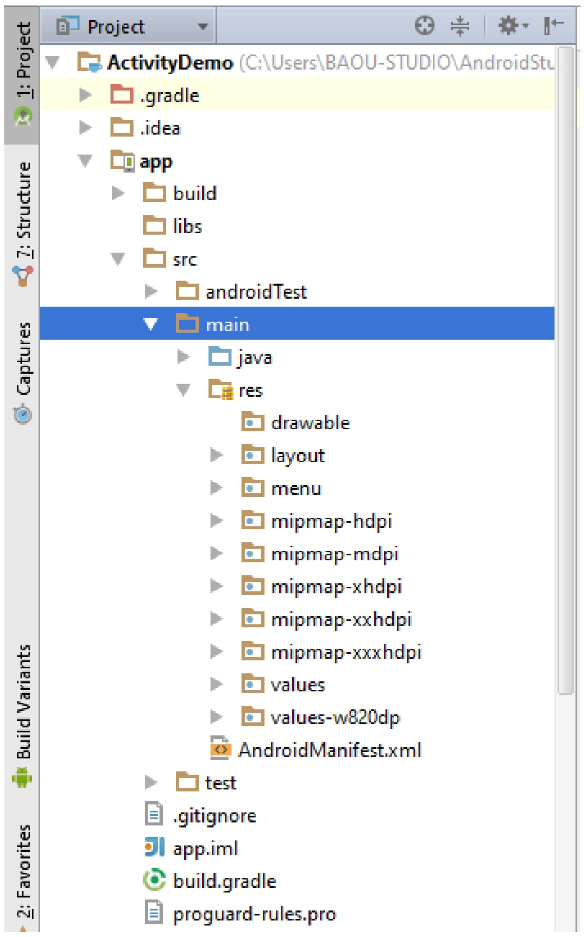
```
Project
ActivityDemo (C:\Users\BAOU-STUDIO\AndroidStu
  .gradle
  .idea
  app
    build
    libs
    src
      androidTest
      main
        java
        res
          drawable
          layout
          menu
          mipmap-hdpi
          mipmap-mdpi
          mipmap-xhdpi
          mipmap-xxhdpi
          mipmap-xxxhdpi
          values
          values-w820dp
        AndroidManifest.xml
      test
    .gitignore
    app.iml
    build.gradle
    proguard-rules.pro
```

Figure-44

File	Meaning
build/	Contains build folders for the specified build variants. Stored in the

File	Meaning
	main application module.
libs/	Contains private libraries. Stored in the main application module.
src/	Contains your stub Activity file, which is stored at src/main/java/<ActivityName>.java. All other source code files (such as .java or .aidl files) go here as well.
androidTest/	Contains the instrumentation tests.
main/jni/	Contains native code using the Java Native Interface (JNI).
main/gen/	Contains the Java files generated by Android Studio, such as your R.java file and interfaces created from AIDL files.
main/assets/	This is empty. You can use it to store raw asset files. For example, this is a good location for textures and game data. Files that you save here are compiled into an .apk file as-is, and the original filename is preserved. You can navigate this directory and read files as a stream of bytes using the AssetManager.
main/res/	Contains application resources, such as drawable files, layout files, and string values in the following directories.
anim/	For XML files that are compiled into animation objects.
color/	For XML files that describe colors.
drawable/	For bitmap files (PNG, JPEG, or GIF), 9-Patch image files, and XML files that describe Drawable shapes or Drawable objects that contain multiple states (normal, pressed, or focused).
mipmap/	For app launcher icons. The Android system retains the resources in this folder (and density-specific folders such as mipmap-xxxhdpi) regardless of the screen resolution of the device where your app is installed. This behavior allows launcher apps to pick the best resolution icon for your app to display on the home screen.
layout/	XML files that are compiled into screen layouts (or part of a screen).
menu/	For XML files that define application menus.
raw/	For arbitrary raw asset files. Saving asset files here is essentially

File	Meaning
	the same as saving them in the assets/ directory. The only difference is how you access them. These files are processed by aapt and must be referenced from the application using a resource identifier in the R class. For example, this is a good place for media, such as MP3 or Ogg files.
values/	For XML files that define resources by XML element type. Unlike other resources in the res/ directory, resources written to XML files in this folder are not referenced by the file name. Instead, the XML element type controls how the resources defined within the XML files are placed into the R class.
xml/	For miscellaneous XML files that configure application components. For example, an XML file that defines a PreferenceScreen, AppWidgetProviderInfo, or Searchability Metadata.
AndroidManifest.xml	The control file that describes the nature of the application and each of its components. For instance, it describes: certain qualities about the activities, services, intent receivers, and content providers; what permissions are requested; what external libraries are needed; what device features are required, what API Levels are supported or required; and others.
.gitignore/	Specifies the untracked files ignored by git.
app.iml/	IntelliJ IDEA module
build.gradle	Customizable properties for the build system. You can edit this file to override default build settings used by the manifest file and also set the location of your keystore and key alias so that the build tools can sign your application when building in release mode. This file is integral to the project, so maintain it in a source revision control system.
proguard-rules.pro	ProGuard settings file.

Table-6

2.5 Types of Modules

Android Studio offers a few distinct types of module:

Android app module: It provides a container for your app's source code, resource files, and app level settings such as the module-level build file and Android Manifest file. When you create a new project, the default module name is "app". In the Create New Module window, Android Studio offers the following types of app modules:

- Phone & Tablet Module
- Wear OS Module
- Android TV Module
- Glass Module

They each provide essential files and some code templates that are appropriate for the corresponding app or device type.

Dynamic feature module: It denotes a modularized feature of your app that can take advantage of Google Play's Dynamic Delivery. For example, with dynamic feature modules, you can provide your users with certain features of your app on-demand or as instant experiences through Google Play Instant.

Library module: It provides a container for your reusable code, which you can use as a dependency in other app modules or import into other projects. Structurally, a library module is the same as an app module, but when built, it creates a code archive file instead of an APK, so it can't be installed on a device.

In the Create New Module window, Android Studio offers the following library modules:

- **Android Library:** This type of library can hold all file types supported in an Android project, including source code, resources, and manifest files. The build result is an Android Archive file or AAR file that can be added as a dependency for your Android app modules.

- **Java Library:** This type of library can contain only Java source files. The build result is a Java Archive or JAR file that can be added as a dependency for your Android app modules or other Java projects.

Google Cloud module: it provides a container for your Google Cloud back end code. It has the required code and dependencies for a Java App Engine back end that uses HTTP, Cloud Endpoints, and Cloud Messaging to connect to your app. You can develop your back end to provide cloud services need by your app.

2.6 Project structure settings

To change various settings for your Android Studio project, open the project structure dialog by clicking File → Project Structure. It contains the following sections:

- **SDK Location:** Sets the location of the JDK, Android SDK, and Android NDK that your project uses.
- **Project:** Sets the version for Gradle and the Android plugin for Gradle, and the repository location name.
- **Developer Services:** Contains settings for Android Studio add-in components from Google or other third parties. See Developer Services, below.
- **Modules:** Allows you to edit module-specific build configurations, including the target and minimum SDK, the app signature, and library dependencies.

2.7 Anatomy of an Android Application

Generally a program is defined in terms of functionality and data, and an Android application is not an exception. It performs processing, show information on the screen, and takes data from a variety of sources.

To Develop Android applications for mobile devices with resource constraint requires a systematic understanding of the application lifecycle. Important terminology for application building blocks terms are Context, Activity, and Intent. This section introduces you with the most important components of Android applications and provides you with a more detailed understanding of how Android applications function and interact with one another.

2.8 Important Android Terminology

The followings are the important terminology used in Android application development.

- **Context:** The context is the essential command for an Android application. It stores the current state of the application/object and all application related functionality can be accessed through the context. Typically you call it to get information regarding another part of your program such as an activity, package, and application.

- **Activity:** It is core to any Android application. An Android application is a collection of tasks, each of which is called an Activity. Each Activity within an application has an exclusive task or purpose. Typically, applications have one or more activities, and the main objective of an activity is to interact with the user.

- **Intent:** Intent is a messaging object which can be used to request an action from another app component. Each request is packaged as Intent. You can think of each such request as a message stating intent to do something. Intent mainly used for three tasks 1) to start an activity, 2) to start a service and 3) to deliver a broadcast.

- **Service:** Tasks that do not require user interaction can be encapsulated in a service. A service is most useful when the operations are lengthy (offloading time-consuming processing) or need to be done regularly (such as checking a server for new mail).

2.9 Basic Android API Packages

Application program interface (API) is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact and APIs are used when programming graphical user interface (GUI) components. Android offers a number of APIs for developing your applications. The following list of core APIs should provide an insight into what's available; all Android devices will offer support for at least these APIs:

API Package	Use
android.util	Provides common utility methods such as date/time manipulation, base64 encoders and decoders, string and number conversion methods, and XML utilities.
android.os	Provides basic operating system services, message passing, and inter-process communication on the device.
android.graphics	Provides low level graphics tools such as canvases, color filters, points, and rectangles that let you handle drawing to the screen directly.
android.text	Provides classes used to render or track text and text spans on the screen.
android.database	Contains classes to explore data returned through a content provider.
android.content	Contains classes for accessing and publishing data on a device.
android.view	Provides classes that expose basic user interface classes that handle screen layout and interaction with the user.
android.widget	The widget package contains (mostly visual) UI elements to use on your Application screen.
android.app	Contains high-level classes encapsulating the overall Android application model.
android.provider	Provides convenience classes to access the content providers supplied by Android.
android.webkit	Provides tools for browsing the web.

Table-7

2.10 Advanced Android API Packages

The core libraries provide all the functionality you need to start creating applications for Android, but it won't be long before you're ready to delve into the advanced APIs that offer the really exciting functionality.

Android hopes to target a wide range of mobile hardware, so be aware that the suitability and implementation of the following APIs will vary depending on the device upon which they are implemented.

API Package	**Use**
android.location	Contains the framework API classes that define Android location-based and related services.
android.media	Provides classes that manage various media interfaces in audio and video.
android.opengl	Provides an OpenGL ES static interface and utilities.
android.hardware	Provides support for hardware features, such as the camera and other sensors.
android.bluetooth	Provides classes that manage Bluetooth functionality, such as scanning for devices, connecting with devices, and managing data transfer between devices. The Bluetooth API supports both "Classic Bluetooth" and Bluetooth Low Energy.
android.net.wifi	Provides classes to manage Wi-Fi functionality on the device.
android.telephony	Provides APIs for monitoring the basic phone information, such as the network type and connection state, plus utilities for manipulating phone number strings.

Table-8

Check your progress-2

a) All application related functionality can be accessed through the _________

b) _____ Package provide classes that manage various media interfaces in audio and video.

c) Tasks that do not require user interaction can be encapsulated in a ________.

d) Application Program Interface is a set of routines, protocols, and tools for building software applications (True/False)

e) Service is a messaging object which can be used to request an action from another app component. (True/False)

f) An Android application is a collection of tasks, each of which is called an Activity (True/False)

2.11 Let us sum up

In this unit you have lean about the structure of Android Project, various types of Android Project files, Android application modules and understand anatomy of an android application. You have also learnt about basic and advanced Application Program Interface.

2.12 Check your Progress: Possible Answers

1-a) Library Modules	1-b) Android Application Modules	1-c) R.java
2-a) Context	2-b) android.media	2-c) Service
2-d) True	2-e) False	2-f) True

2.13 Further Reading

- https://developer.android.com/studio/projects

2.14 Assignment

- Explain Android Project Structure
- Write detailed note on basic and advanced API packages
- Define: Activity, Service, Context, Intent