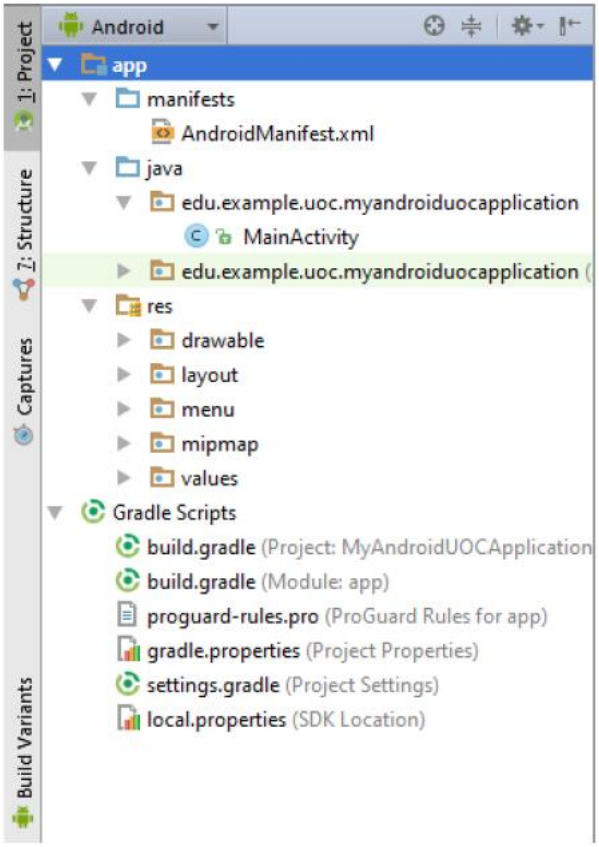
Una aplicación Android está compuesta por una serie de ficheros básicos, la mayoría de los cuales se generan automáticamente al crear una nueva aplicación en Android Studio. En la siguiente imagen se puede ver la estructura básica de una aplicación Android, vista en Android Studio con el modo de ordenar paquetes "Android", que se selecciona en el desplegable superior:



Los siguientes ficheros son los que se utilizarán principalmente para crear una aplicación Android:

- AndroidManifest.xml
- build.gradle
- Layouts XML
- Actividades y fragmentos
- Ficheros de recursos

El resto de los ficheros son utilizados por Android Studio para configurar otros parámetros tanto de Android (localización del SDK), como de configuración de Gradle y de compilación. También se crean automáticamente unas clases de Test, que permitirán testear ciertas funcionalidades de la aplicación basándose en JUnit.

1.1. Android Manifest

El fichero AndroidManifest.xml se encuentra en el directorio raíz de la aplicación y es obligatorio en todas las aplicaciones Android, ya que en él se encuentra la información esencial de cada aplicación Android.

En la siguiente imagen de ejemplo se puede ver el Android Manifest que se genera por defecto al crear una nueva aplicación con Android Studio:

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.example.uoc.myandroiduocapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MyAndroidUOCApplication"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="MyAndroidUOCApplication"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

En este fichero de tipo XML es donde se declarará la siguiente información de la aplicación:

- El **identificador único de la aplicación**, el cual es representado por el paquete Java de la aplicación y se introduce en el atributo "package" de la etiqueta principal "manifest" del AndroidManifest.xml.

  Por ejemplo, en el siguiente código se define el identificador único "edu.example.uoc.myandroiduocapplication" de una aplicación:

  ```xml
  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="edu.example.uoc.myandroiduocapplication">
  ```

- Los **componentes** de la aplicación. Todos los componentes de la aplicación, ya sean **actividades**, **servicios**, ***broadcast receivers*** o ***content providers*** deben ser definidos en el Manifest para que puedan ser lanzados por el sistema Android cuando estos son llamados en algún punto de la aplicación. Estos se definen con el siguiente formato e incluyen las siguientes etiquetas:

```xml
<activity>
    <intent-filter>
      <action />
      <category />
      <data />
    </intent-filter>
    <meta-data />
  </activity>
  <service>
    <intent-filter> . . . </intent-filter>
    <meta-data/>
  </service>
  <receiver>
    <intent-filter> . . . </intent-filter>

    <meta-data />
  </receiver>
  <provider>
    <grant-uri-permission />
    <meta-data />
    <path-permission />
  </provider>
```

1.2. Build.gradle

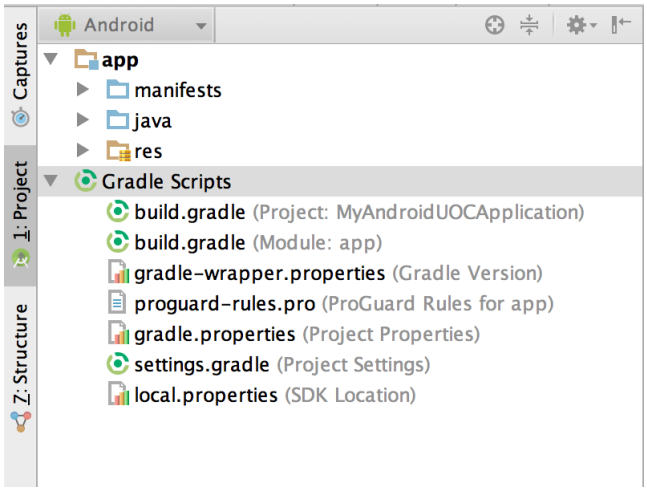
El fichero **build.gradle** es el fichero más importante para compilar una aplicación con **Gradle**. Se trata de una herramienta Open Source con licencia Apache Software License (ASL) que se empezó a utilizar en proyectos Android tras la publicación de Android Studio;

actualmente es la herramienta oficial de Google para la compilación de proyectos Android, sustituyendo a la anterior herramienta Ant, que se utilizaba junto a Eclipse.

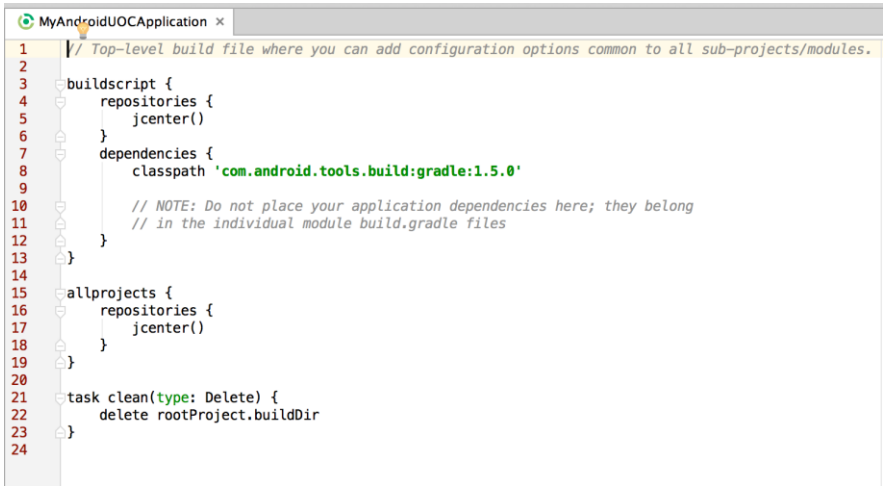
Gradle es una herramienta muy potente, con muchas posibilidades y que nos va a facilitar la automatización de la construcción de proyectos Android. Esta herramienta permite, entre otras cosas, compilar, testear, empaquetar y desplegar las aplicaciones Android. Se trata de una herramienta que utiliza compilaciones incrementales, por lo que no compilará cada vez todo el código entero, sino que solo lo compilará en caso de que se haya producido un cambio en el código fuente desde la última compilación. Por otra parte, también soporta múltiples proyectos, lo que permite a sus usuarios compilar aplicaciones con otros proyectos como librerías.

Como se ha comentado previamente, Gradle es la herramienta oficial de Google para proyectos Android en Android Studio. Por tanto, al descargarse Android Studio, este ya se descargará también. Para una configuración simple en la que se necesite únicamente compilar, lanzar aplicaciones y empaquetar, lo único que se tendrá que hacer para cada proyecto Android es configurar su fichero build.gradle, el cual ya viene preconfigurado al crear un proyecto nuevo.

Tal y como se muestra en la siguiente imagen, en Android Studio, debajo de la carpeta de ficheros del proyecto, se encuentra la sección de configuración de Gradle (Gradle Scripts):



El primer fichero build.gradle es el que se encarga de la compilación del proyecto Android entero. En caso de que tengamos configuraciones que sean comunes para varios módulos o subproyectos de la aplicación, se podrán definir en este fichero. Por ejemplo, el repositorio utilizado para buscar las librerías incluidas en el proyecto, o la dependencia de la librería de Gradle, se definen en este fichero. En la siguiente imagen se puede ver un fichero de ejemplo, el cual se creará automáticamente al crear un nuevo proyecto en Android Studio:



El siguiente fichero build.gradle es el que llevará la configuración de la aplicación. En caso de que el proyecto Android que se esté realizando no tenga más módulos o subproyectos, no habrá más ficheros build.gradle. Si hubiera más módulos, se diferenciarán por el nombre entre paréntesis que pone Android Studio y cada fichero estará en el directorio raíz del módulo o subproyecto incluido. En este caso, únicamente hay un módulo y, al ser el que se crea por defecto, se llama "app". Este es el nombre por defecto que pone Android Studio al módulo principal de una aplicación.

En este fichero, para la compilación de una aplicación, será necesario definir lo siguiente:

- La versión del SDK de Android con la que se compilará la aplicación: **compileSdkVersion**. Esta es la versión con la que se compila la aplicación, no la mínima y máxima versión de Android en la que la aplicación puede correr.
- La versión de las herramientas de Android: **buildToolsVersion**.
- El identificador único de la aplicación, el mismo que se ha definido en el Android Manifest: **applicationId**.
- La versión mínima, máxima y la objetivo de Android: **minSdkVersion**, **maxSdkVersion**, **targetSdkVersion**. La versión objetivo de Android es la versión en la que se ha probado la aplicación y con la cual funciona todo correctamente. Lo ideal es que la versión objetivo de Android sea la última versión de Android para compilar el código con la última versión y probar la aplicación con las últimas funcionalidades de Android.
- La versión de la aplicación: **versionCode**. Esta versión se tendrá que ir aumentando cada vez que se libera una versión nueva de la aplicación.
- El nombre de la versión de la aplicación: **versionName**. Esta es la versión que verán los usuarios que se descarguen la aplicación.
- Las dependencias o librerías de la aplicación: **dependencies**. En caso de que se incluyan las dependencias de la aplicación como ficheros "JAR" o añadiendo las librerías a la carpeta de "libs" de la aplicación, se deberá incluir la siguiente línea:

  compile fileTree(dir: 'libs', include: ['*.jar'])

- Y en caso de que las dependencias de la aplicación se descarguen directamente por Gradle desde el repositorio de librerías, se deberá incluir la siguiente línea para cada librería que se quiera utilizar:

  compile 'identificador.aplicacion:nombre:version'

En la siguiente imagen, se puede ver un fichero build.xml de ejemplo, generado automáticamente por Android Studio al crear un nuevo proyecto:

app ×

```
1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 23
5      buildToolsVersion "23.0.2"
6
7      defaultConfig {
8          applicationId "edu.example.uoc.myandroiduocapplication"
9          minSdkVersion 14
10         targetSdkVersion 23
11         versionCode 1
12         versionName "1.0"
13     }
14     buildTypes {
15         release {
16             minifyEnabled false
17             proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18         }
19     }
20 }
21
22 dependencies {
23     compile fileTree(dir: 'libs', include: ['*.jar'])
24     testCompile 'junit:junit:4.12'
25     compile 'com.android.support:appcompat-v7:23.1.1'
26     compile 'com.android.support:design:23.1.1'
27 }
28
```

En la imagen de ejemplo, la aplicación se compila con la version 23 de la API de Android (versión 6.0) y la versión 23.0.2 de las herramientas de Android. Su configuración por defecto incluye el identificador de la aplicación "edu.example.uoc.myandroiduocapplication", la mínima versión de Android en la que funcionará es la 14 (versión 4.0), la aplicación ha sido testeada hasta la versión 23 y su número de versión es el 1, el cual se mostrará como "1.0".

Los **buildTypes** permitirán configurar diferentes tipos de compilaciones, es decir, dependiendo de la configuración de compilación que se seleccione en el apartado "Build variants", compilará una versión u otra. Esto puede ser muy útil cuando por ejemplo necesitamos compilar dos versiones de una aplicación, una que conecte con un servidor de pruebas y otra que conecte con el servidor definitivo de producción. En este caso, se pueden definir dentro de los buildTypes dos tipos que definan una variable (**buildConfigField**), que será verdadera o falsa según si estamos en modo de compilación "de pruebas" o compilación de "producción". En el caso de la imagen, se define una versión especial de tipo "release", la cual podrá estar ofuscada con proguard.

Por último, se definen las dependencias de la aplicación. En este caso, se incluyen como dependencias todas las que estén en la carpeta "libs" y que acaben en ".jar". Para testear la aplicación, la librería JUnit, y por último dos librerías de soporte de Android:

appcompat-v7 y design.

En conclusión, al preparar una aplicación para compilar y empaquetar, es importante especificar la versión mínima del SDK de Android que soporta (minSdkVersion) y la versión objetivo (targetSdkVersion), así como el número de versión que la aplicación tendrá en Google Play (versionCode y versionName). Por otro lado, también se definirán en este fichero las librerías que se utilizarán en la aplicación (*dependencies*).

1.3. Layouts XML

Los **layouts XML** son una serie de ficheros XML que permiten configurar la apariencia de la aplicación. En Android, los *layouts* se pueden configurar de dos maneras: definiendo un XML con la estructura de la vista o, directamente en tiempo de ejecución, creando vistas u objetos de diferentes tipos desde las actividades o *fragments*. Siempre que sea posible se recomienda declarar las vistas desde el XML, ya que es la forma más sencilla, rápida y ordenada de crear vistas u objetos, separando así la parte de presentación de la aplicación de la lógica.

Según las vistas que se vayan a crear, existen diferentes tipos de *layouts* que nos permitirán configurar todo tipo de diseños de pantalla. Cada *layout* deberá tener un elemento raíz, el cual deberá ser de tipo **View** o **ViewGroup**, o bien extender a uno de estos dos tipos. Una vez tenemos el elemento raíz, se podrán colocar tantos elementos hijo como sean necesarios para configurar la vista.

Por ejemplo, en la siguiente imagen se puede ver un *layout* muy simple que incluye un elemento raíz, que en este caso es de tipo RelativeLayout, y un elemento hijo, que es de tipo TextView. Con estos elementos se creará una vista muy simple que mostrará el texto "Hello world!" con un *padding* de 16 dp:

content_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

</RelativeLayout>
```

Si se pulsa sobre la pestaña de la derecha que pone "Preview", se podrá ver el resultado de este *layout* tal y como se mostraría en diferentes dispositivos. Esta opción únicamente aparece disponible cuando estamos editando un *layout* u otro elemento visual.

En la siguiente imagen se muestra la visualización previa del *layout* anterior, vista en un dispositivo Nexus 4 y en formato vertical:

Estos ficheros XML se encuentran dentro de la carpeta **layout**, incluida en el directorio **res** de una aplicación Android. Como en Android hay muchos tipos de dispositivos con

diferentes tipos de pantallas y resoluciones, en ocasiones con un único *layout* para todas las pantallas no nos será suficiente. Por ello, las carpetas del directorio **res** pueden ser duplicadas con diferentes sufijos que permitirán crear *layouts* distintos para todos los diferentes tipos de dispositivos.

Por ejemplo, si los *layouts* de nuestra aplicación únicamente difieren por pantallas más pequeñas o más grandes, las carpetas *layouts* se llamarían de la siguiente manera y en cada carpeta se volvería a crear el fichero layout XML que varía según si la pantalla es pequeña, normal, grande o muy grande:

- **layout-small**
- **layout-normal**
- **layout-large**
- **layout-xlarge**

Otra manera de clasificar los *layouts* en Android es dependiendo de su densidad de pantalla. Por tanto, las carpetas de *layout* se dividirían según si la densidad de pantalla es baja (*low*), media (*medium*), alta (*high*), muy alta (*extra high*), etc.:

- **layout-ldpi**
- **layout-mdpi**
- **layout-hdpi**
- **layout-xhdpi**
- **layout-xxhdpi**
- **layout-xxxhdpi**

Por otra parte, si el *layout* cambia según la orientación del dispositivo, se crearán dos carpetas: una para horizontal (*landscape*) y otra para vertical (*portrait*):

- **layout-land**
- **layout-port**

Estas configuraciones se pueden combinar entre sí, por ejemplo, para una densidad alta de pantalla en orientación horizontal, la carpeta se llamaría "**layout-hdpi-landscape**".

Por último, otra forma interesante para tener diferentes layouts es por idioma. Esto puede resultar útil cuando la aplicación vaya a funcionar en diferentes idiomas y alguno necesite un *layout* específico. Por ejemplo, se podrían utilizar los siguientes *layouts*:

- **layout-ar**: *layout* para arábigo
- **layout-es**: *layout* para español
- **layout-ldrtl**: *layout* específico para las lenguas que se escriben de derecha a izquierda (*layout direction right to left*)
- **layout-en-rUS**: *layout* para inglés de Estados Unidos Estos sufijos son aplicables al resto de las carpetas de recursos de Android

Sobre los tipos de layouts hablaremos con más precisión más adelante.

1.4. Actividades y fragmentos

Las actividades y fragmentos son los elementos que nos permiten crear las pantallas que se visualizarán en el dispositivo móvil.

Por tanto, las actividades son una de las partes más importantes de una aplicación Android, ya que nos permiten ver las pantallas de la aplicación.

Las actividades se componen de dos partes: una parte de lógica y una parte gráfica. La parte de lógica se corresponde al fichero ".java", donde se realizan todas las acciones de la pantalla. En cambio, la parte gráfica se corresponde al *layout* XML que se ha explicado previamente, donde se define toda la forma visual de la pantalla.

Los fragmentos, en cambio, son como pseudoactividades, ya que no pueden existir sin una actividad, pero de la misma manera que la actividad, estos permiten crear las pantallas de una aplicación.

Es conveniente utilizar los fragmentos en varios casos y siempre dentro de otra actividad:

- Cuando el código que se introduzca en el fragmento vaya a reusarse, ya que los fragmentos se pueden instanciar desde distintas actividades.
- Cuando se utilicen interfaces con diseños dinámicos, como por ejemplo pestañas, paginación, diseños master-detail, etc.
- Cuando varias pantallas estén relacionadas entre sí y tengan elementos en común, entonces también es conveniente utilizar varios fragmentos con una actividad que contenga los elementos en común.

Las actividades y los fragmentos se encuentran en el directorio “java” de la aplicación y se distribuyen en las carpetas correspondientes al identificador de la aplicación. Así, por ejemplo, si el identificador de una aplicación es “com.dam.chari.calculadora”, las clases Java de la aplicación se encontrarán en el directorio java/com/dam/chari/calculadora.

Una de las cosas más importantes a la hora de realizar una aplicación Android es saber en qué momento se debe usar una actividad y en qué momento se debe usar un fragmento.

1.5 Ficheros de recursos

Por último, dentro de la carpeta “res” de una aplicación Android se encuentran todos los recursos de esta.

Las carpetas de recursos que puede contener son las siguientes:

- animator: permite incluir ficheros XML que definen propiedades de las animaciones.
- anim: permite incluir ficheros XML de animaciones.
- color: permite incluir ficheros XML que definen listas de estados de colores, es decir, un color que cambia según el estado del elemento al que se aplica.
- drawable: permite incluir ficheros de imágenes bitmap, *nine-patches* (imágenes que pueden adaptar el tamaño), formas, animaciones con imágenes, etc.
- mipmap: permite definir el icono de lanzamiento de la aplicación para diferentes densidades de pantalla. Para añadir una imagen para cada densidad, se tienen que crear las diferentes carpetas de densidades de mipmap: mipmap-ldpi, mipmap-mdpi, mipmap-hdpi, etc.
- layout: en esta carpeta se incluyen los ficheros de los layouts XML, explicados anteriormente.
- menu: incluye los ficheros XML para todo tipo de menús.
- raw: permite incluir aquí ficheros de diferentes formatos accediendo a ellos a través de su identificador. Este tipo de ficheros también pueden ser guardados en la carpeta Assets.
- values: permite guardar ficheros XML que guardan valores. Algunos de los ficheros de valores más usados que se guardan aquí son los siguientes:
    - **values.xml**: para los *strings* de valores (usado sobre todo para idioma).
    - **colors.xml**: para los valores de los colores utilizados en la aplicación.
    - **dimens.xml**: para los valores de dimensiones utilizados en la aplicación.
    - **styles.xml**: para los estilos utilizados.
    - **arrays.xml**: para valores de *arrays*.
    - **xml**: permite guardar cualquier otro tipo de fichero XML.