

## INDICE

---

1.1	COMPONENTES DE UNA APLICACIÓN	1
1.2	ACTIVITIES	3
1.3	CREAR EL PROYECTO: HELLO WORLD	4
1.4	FICHEROS Y CARPETAS DE UN PROYECTO ANDROID	4
1.5	SOPORTE MÚLTIPLES PANTALLAS.	11

## 1.1 Componentes de una aplicación

Los componentes de la aplicación son **bloques de creación esenciales de una aplicación** para Android. Cada componente es un punto diferente a través del cual el sistema puede ingresar a tu aplicación. No todos los componentes son puntos de entrada reales para el usuario y algunos son dependientes entre sí, pero cada uno existe como entidad individual y cumple un rol específico; cada uno es un bloque de creación único que ayuda a definir el comportamiento general de tu aplicación. En esta sección vamos a realizar una descripción inicial de algunos de los más importantes. A lo largo del curso trabajaremos con algunos de ellos (no todos) y se describirán con más detalle las clases Java que implementan.

### Vista (View)

Las vistas son los elementos que componen la interfaz de usuario de una aplicación: por ejemplo, un botón o una entrada de texto. Todas las vistas van a ser objetos descendientes de la clase View, y por tanto, pueden ser definidas utilizando código Java. Sin embargo, lo habitual será definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero. Esta forma de trabajar es muy similar a la definición de una página web utilizando código HTML..

### Layout

Un *layout* es un conjunto de vistas agrupadas de una determinada forma. Vamos a disponer de diferentes tipos de layouts para organizar la vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los *layouts* también son objetos descendientes de la clase View. Igual que las vistas, los *layouts* pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML.

### Activity

Un *layout* Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, coloquialmente conocidos como pantallas de la aplicación. En Android cada uno de estos elementos, o pantallas, se conoce como actividad. Su función principal es la creación de la interfaz de usuario. Una aplicación suele necesitar varias actividades

para crear la interfaz de usuario. Las diferentes actividades creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común. Una actividad se define en una clase descendiente de Activity y utiliza un layout para que defina su apariencia.

## Fragmentos (Fragment)

La llegada de las tabletas trajo el problema de que las aplicaciones de Android ahora deben soportar pantallas más grandes. Si diseñamos una aplicación pensada para un dispositivo móvil y luego la ejecutamos en una tableta, el resultado no suele resultar satisfactorio.

Para ayudar al diseñador a resolver este problema, en la versión 3.0 de Android aparecen los *fragments*. Un *fragment* está formado por la unión de varias vistas para crear un bloque funcional de la interfaz de usuario. Una vez creados los fragments, podemos combinar uno o varios *fragments* dentro de una actividad, según el tamaño de pantalla disponible.

Su uso puede ser algo complejo, por lo que recomienda primero dominar los conceptos de activity, vista y layout antes.

## Servicio (Service)

Un servicio es un proceso que se ejecuta “detrás”, sin la necesidad de una interacción con el usuario. Es algo parecido a un *demonio* en Unix o a un *servicio* en Windows. Se utilizan cuando queramos tener en ejecución un código de manera continua, aunque el usuario cambie de actividad. En Android disponemos de dos tipos de servicios: servicios locales, que son ejecutados en el mismo proceso, y servicios remotos, que son ejecutados en procesos separados.

## Intención (Intent)

Una intención representa la voluntad de realizar alguna acción; como realizar una llamada de teléfono, visualizar una página web. Se utiliza cada vez que queramos:

- Lanzar una actividad
- Lanzar un servicio
- Enviar un anuncio de tipo broadcast
- Comunicarnos con un servicio

Los componentes lanzados pueden ser internos o externos a nuestra aplicación. También utilizaremos las intenciones para el intercambio de información entre estos componentes.

## Receptor de anuncios (Broadcast Receiver)

Un receptor de anuncios recibe anuncios broadcast y reacciona ante ellos. Los anuncios broadcast pueden ser originados por el sistema (por ejemplo: Batería baja, Llamada entrante) o por las aplicaciones. Las aplicaciones también pueden crear y lanzar nuevos tipos de anuncios broadcast. Los receptores de anuncios no disponen de interfaz de usuario, aunque pueden iniciar una actividad si lo estiman oportuno.

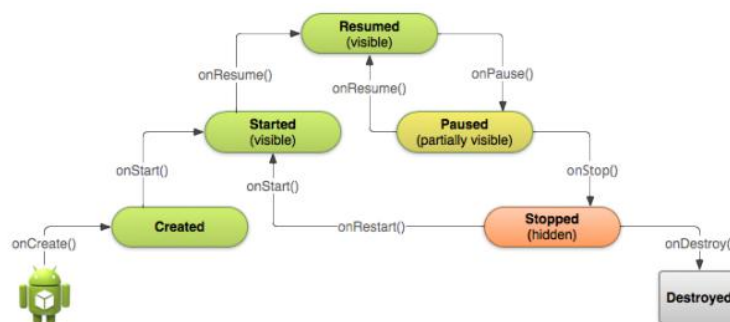
## Proveedores de Contenido (Content Provider)

En muchas ocasiones las aplicaciones instaladas en un terminal Android necesitan compartir información. Android define un mecanismo estándar para que las aplicaciones puedan compartir datos sin necesidad de comprometer la seguridad del sistema de ficheros. Con este mecanismo podremos acceder a datos de otras aplicaciones, como la lista de contactos, o proporcionar datos a otras aplicaciones.

## 1.2 Activities

En la unidad anterior ya explicamos qué era una **Activity**. Más adelante las veremos con más detalle. A modo de recordatorio:

- Una **Activity** es una pantalla única con la que el usuario puede interactuar a través de los elementos disponibles en ella.
- Una **aplicación** constará de una o más actividades.
- Incluso se puede pasar de la **Activity** de una aplicación a cualquier **Activity** de otra aplicación. Por ejemplo, al usar WhatsApp y acceder a los detalles de un contacto.
- Se puede llamar a una **Activity** cuando se inicia una aplicación, desde otra aplicación o cuando se recupera la aplicación.
- Una **Activity** puede ser destruida por la propia aplicación, cuando se presiona el botón **Back** del dispositivo o por el sistema porque está en la pila de aplicaciones en segundo plano y se necesitan los recursos que se están consumiendo.
- Las **Activities** tienen un ciclo de vida como se muestra de forma sencilla en la siguiente imagen:



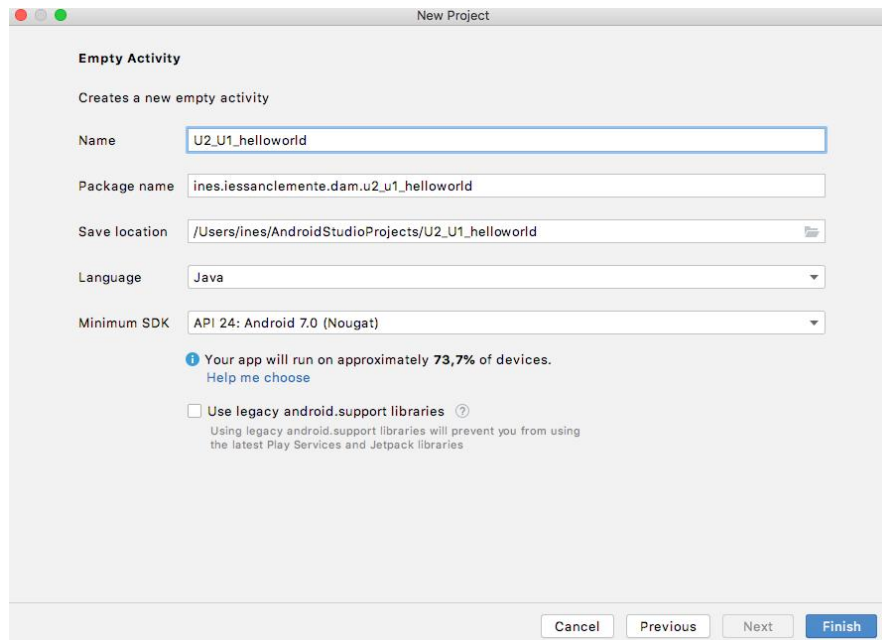
### Referencias:

- <http://developer.android.com/reference/android/app/Activity.html>
- <http://developer.android.com/guide/componentsactivities.html>
- <http://developer.android.com/training/basics/activity-lifecycle/index.html>

## 1.3 Crear el proyecto: Hello World

En la siguiente sección se describirán las carpetas y archivos más esenciales que componen un proyecto de Android. Empezaremos creando un proyecto: **Hello World**, y estudiaremos las partes en las que se divide. Los proyectos se nombrarán y numerarán de la siguiente manera: **U2\_XX\_ApplicationName**, para facilitar el seguimiento. U2 es la unidad 2 y XX el número de proyecto dentro de la unidad.

Según lo visto en la unidad 1, creamos un proyecto a partir de la plantilla Empty Activity, donde luego vamos a crear la actividad Hello World!!!.

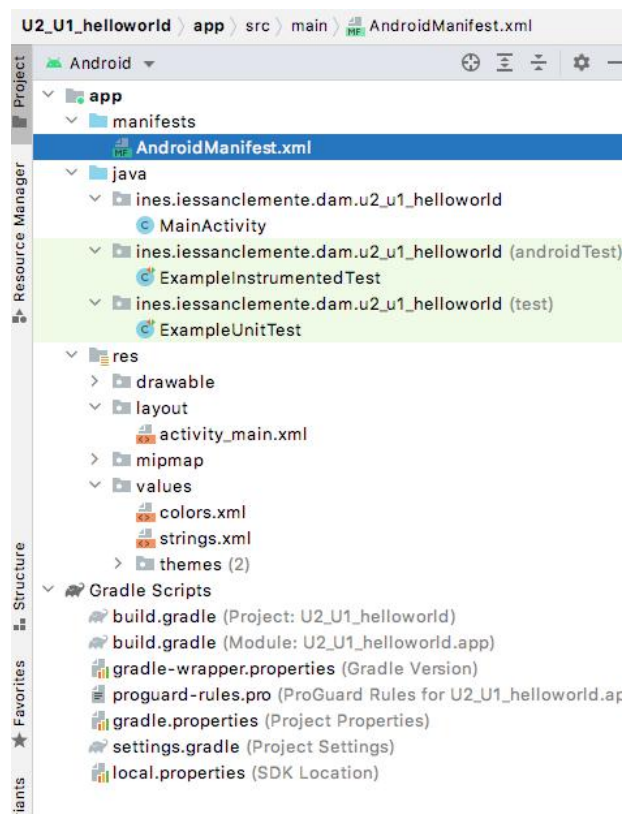


## 1.4 Ficheros y carpetas de un proyecto Android

Lo primero que debemos conocer es que un proyecto en Android Studio puede contener varios módulos. Cada módulo corresponde a una aplicación o ejecutable diferente. Disponer de varios módulos en un mismo proyecto es muy útil cuando queremos crear varias versiones de nuestra aplicación, por ejemplo para dispositivo móvil, Wear, TV, Things, etc. También si queremos crear varias versiones de **nuestra** aplicación con nivel mínimo de SDK diferentes.

En este tema solo vamos a desarrollar aplicaciones para móviles, por lo que no vamos a necesitar solo un módulo. Este módulo ha sido creado con el nombre **app**.

Cada módulo en Android está formado por un descriptor de la aplicación (**manifests**), el código fuente en Java (**java**), una serie de ficheros con recursos (**res**) y ficheros para construir el módulo (**Gradle Scripts**). Cada elemento se almacena en una carpeta específica, indicada entre paréntesis. Aprovecharemos el proyecto que acabamos de crear para estudiar la estructura de un proyecto en Android Studio. No debemos asustarnos con el exceso de información. Más adelante se dará más detalles sobre la finalidad de cada fichero.

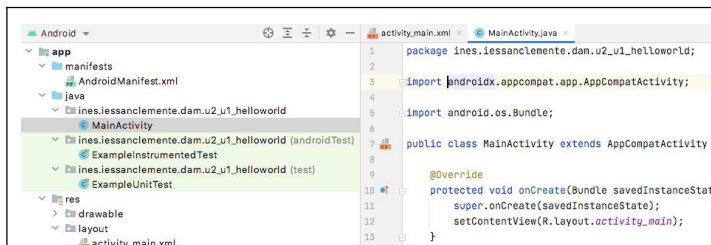


## AndroidManifest.xml

Este fichero describe la aplicación Android. Aquí se define su nombre, paquete, icono, estilos, etc. Se indican las *actividades*, las *intenciones*, los *servicios*, y los *proveedores de contenido* de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica el paquete Java, la versión de la aplicación, etc.

## java

Carpeta que contiene el código fuente de la aplicación. Como puedes observar los ficheros Java se almacenan en carpetas según el nombre de su paquete.



**MainActivity:** Clase con el código de la actividad inicial (inicia la pantalla principal y el menú).

La clase **U2\_01\_MainActivity** hereda de la clase **AppCompatActivity** sus atributos y métodos. En este caso, se sobrescribe el método **onCreate**, que es el primero en ejecutarse cuando se lanza una actividad (ver imagen en sección 1.2 de los estados de una actividad). Este método recibe un parámetro **Bundle**, en el que recibe el estado de la actividad:

- (-Null): porque acabamos de empezar
- o **valores del estado** anterior si acaba de detenerse. Mirar la imagen del ciclo de vida).

Llama al método **onCreate** de la clase principal.

A continuación se muestra en pantalla el contenido del layout: **activity\_main**. Para ello utilizar la clase **R** (de recursos) como veremos a continuación.

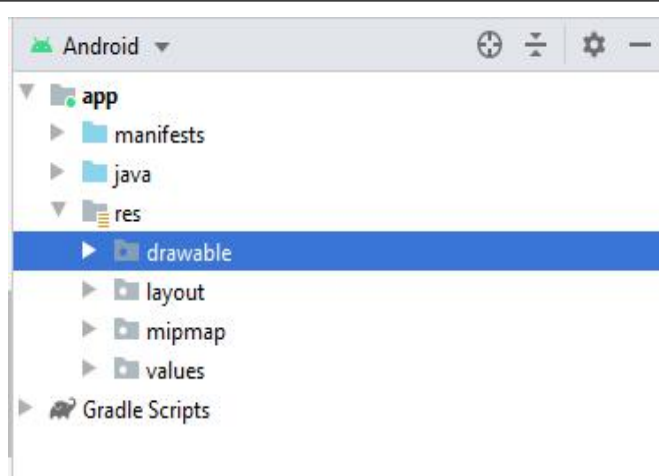
<https://developer.android.com/reference/android/os/Bundle.html>

[https://developer.android.com/reference/android/app/Activity.html#setContentView\(int\)](https://developer.android.com/reference/android/app/Activity.html#setContentView(int))

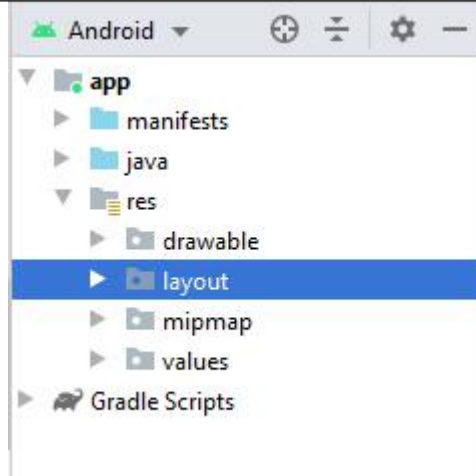
## res

**Carpeta res** (Recursos/Resources): almacena todos los archivos de recursos necesarios para la aplicación: imágenes, videos, ficheros xml, animaciones, ficheros para internacionalización, etc. A continuación se describen los distintos recursos (subcarpetas) que puede contener (no solo son los cuatro creados por defecto: drawable, layout, mipmap, values).

### /res



**/res/drawable:** En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.



**layout:** Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas web.

```

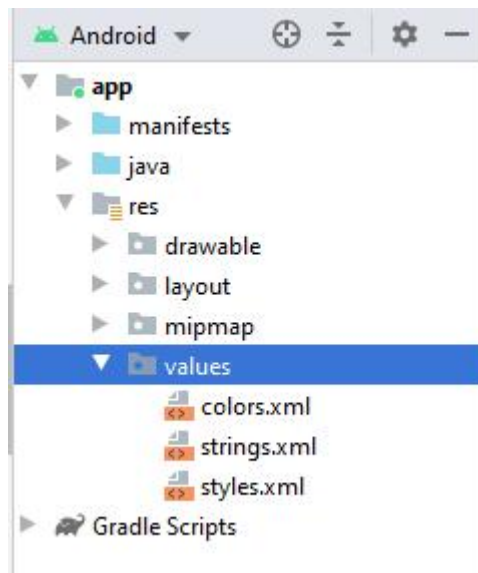
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

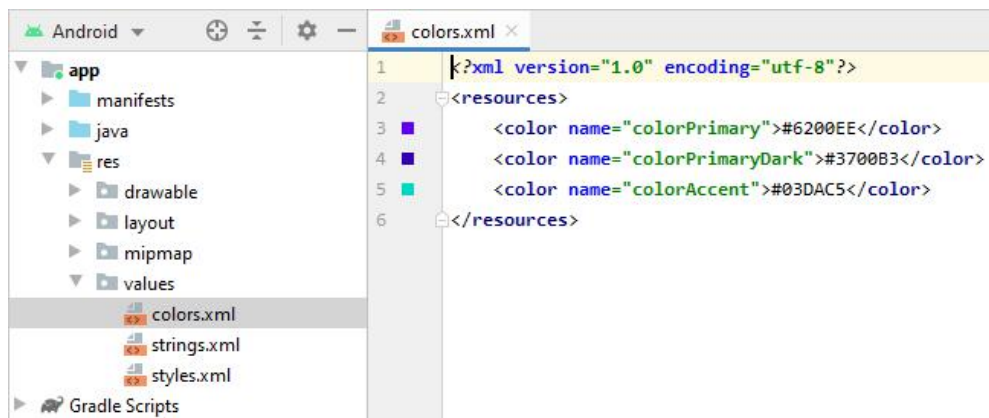
</androidx.constraintlayout.widget.ConstraintLayout>

```

Este es el contenido del archivo XML de diseño.

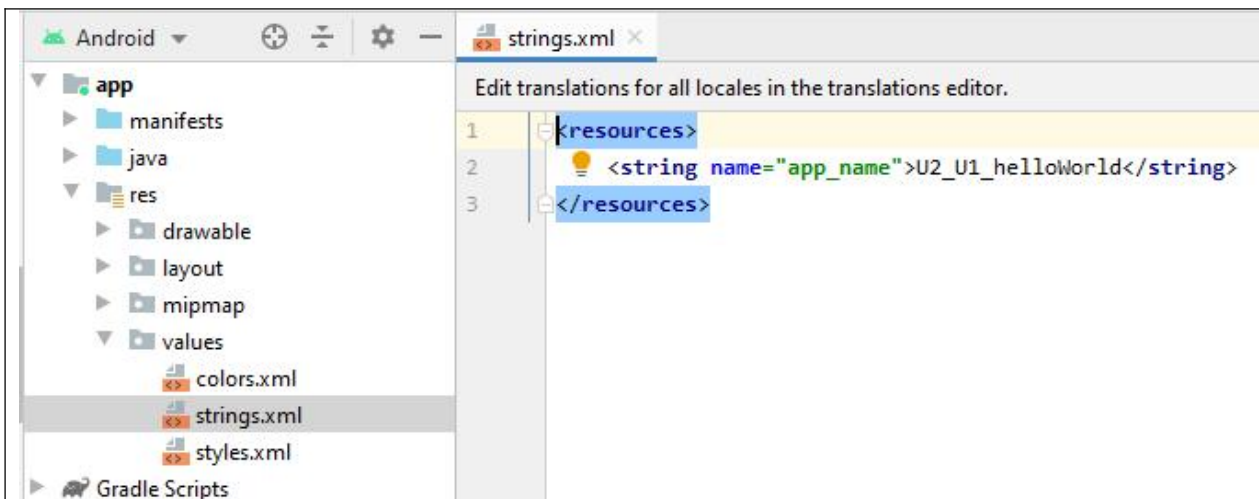


**/res/values:** También utilizaremos ficheros XML para indicar valores usados en la aplicación, de esta manera podremos cambiarlos desde estos ficheros sin necesidad de ir al código fuente.

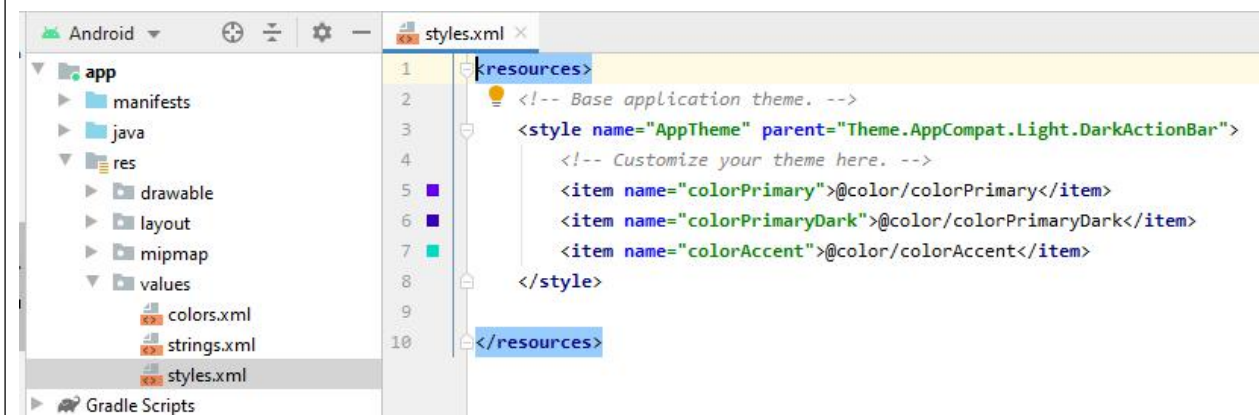


En **colors.xml** se definen los tres colores primarios de la aplicación.



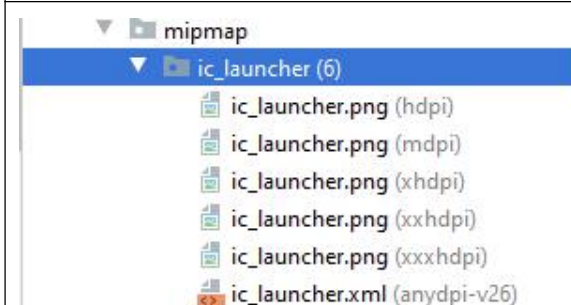


En el fichero **strings.xml**, tendrás que definir todas las cadenas de caracteres de tu aplicación. Creando recursos alternativos resultará muy sencillo traducir una aplicación a otro idioma.



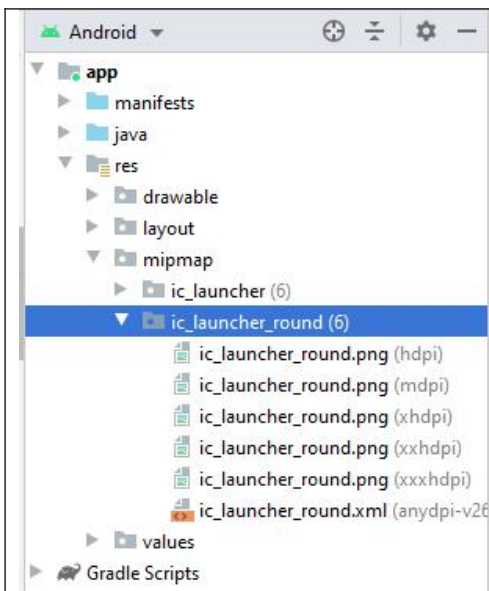
Finalmente en **styles.xml**, podrás definir los estilos y temas de tu aplicación.

En **dimens.xml**, se pueden definir dimensiones como el margen por defecto o el ancho de los botones. En este ejemplo no existe el fichero.



**/res/mipmap:** En una carpeta guardaremos el icono de la aplicación. En el proyecto se ha incluido el fichero **ic\_launcher.png** que será utilizado como icono de la aplicación. Observa cómo este recurso se ha añadido en seis versiones diferentes. Usaremos un sufijo especial si queremos tener varias versiones de un recurso, de forma que solo se cargue al cumplirse una determinada condición. Por ejemplo: (**hdpi**) significa que solo ha de cargar los recursos contenidos en esta carpeta cuando el dispositivo donde se instala la aplicación tenga una densidad gráfica alta (180- dpi); (**mdpi**) se utilizará con densidad gráfica alta (180- dpi). Si pulsas sobre las diferentes versiones del recurso, observarás como se trata del mismo icono pero con más o menos resolución de forma que, en función de la densidad gráfica del dispositivo, se ocupe un tamaño similar en la pantalla.





El fichero **ic\_launcher\_round.png** es similar pero para cuando se quieren usar iconos redondos.

**raw:** Ficheros adicionales que no se encuentran en formato XML.

**menu:** Ficheros XML con los menús de cada actividad. En el proyecto no hay ningún menú por lo que no se muestra esta carpeta.

**anim:** Ficheros XML con animaciones de vistas

**animator:** Ficheros XML con animaciones de propiedades

**xml:** otros ficheros XML requeridos por la aplicación

## Gradle Scripts

En esta carpeta se almacenan una serie de ficheros *Gradle* que permiten compilar y construir la aplicación. Observar como algunos hacen referencia al módulo *app* y el resto son para configurar todo el proyecto. El fichero más importante es *build.gradle (Module:app)* que es donde se configuran las opciones de compilación del módulo:

```
plugins {
    id 'com.android.application'
}

android {
    compileSdk 31

    defaultConfig {
        applicationId "ines.iessanclemente.dam.u2_u1_helloworld"
        minSdk 24
        targetSdk 31
    }
}
```

```
versionCode 1
versionName "1.0"

testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
}

buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
}
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
}

dependencies {

    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'
    testImplementation 'junit:junit:4.+ '
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

El primer parámetro que podemos configurar es **compileSdkVersion** que nos permite definir la versión del sdk con la que compilamos la aplicación. Las nuevas versiones no solo añaden funcionalidades al API, también añaden mejoras en los procesos. Por ejemplo, a partir de la versión 3.0 (API 11) solo se permite el acceso a Internet desde un hilo auxiliar[2]. **applicationId** suele coincidir con el nombre del paquete Java creado para la aplicación. Se utiliza como identificador único de la aplicación, de forma que no se permite instalar una aplicación si ya existe otra con el mismo id. **minSdkVersion** especifica el nivel mínimo de API que requiere la aplicación. Es un parámetro de gran importancia, la aplicación no podrá ser instalada en dispositivos con versiones anteriores y solo podremos usar las funcionalidades del API hasta este nivel (con excepción de las librerías de compatibilidad). **targetSdkVersion** indica la versión más alta con la que se ha puesto a prueba la aplicación. Cuando salgan nuevas versiones del SDK tendrás que comprobar la aplicación con estas versiones y actualizar el valor. **versionCode** y **versionName** indica la versión de tu aplicación. Cada vez que publiques una nueva versión incrementa en uno el valor de versionCode y aumenta el valor de versionName según la importancia de la actualización. Si es una actualización menor el nuevo valor podría ser "1.1" y si es mayor "2.0".

Dentro de **buildTypes** se añaden otras configuraciones dependiendo del tipo de compilación que queramos (*release* para distribución, *debug* para depuración, etc.).

Un apartado importante es el de **dependencias**. En él has de indicar todas las librerías que han de ser incluidas en nuestro proyecto. Si necesitas usar alguna librería de compatibilidad adicional has de incluirla aquí.

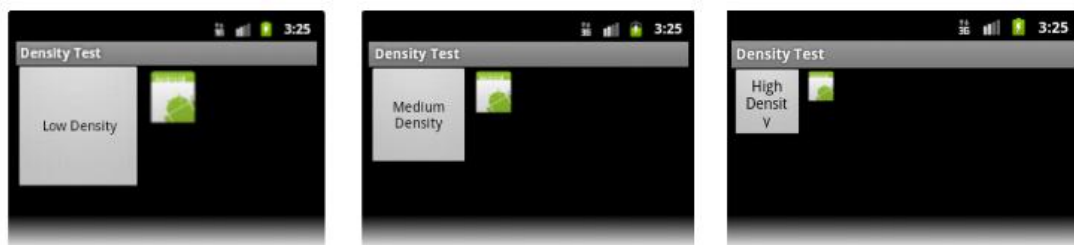
Nota: recomiendo se visualice el videotutorial (Estructura de un proyecto Android) indicado en las referencias. Ayudará a entender mejor la estructura de un proyecto Android.

## Referencias:

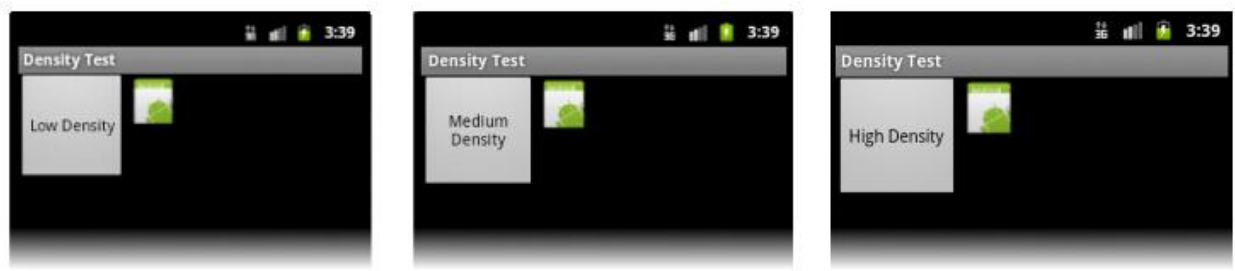
VideoTutorial: [Estructura de un proyecto Android](#)

## 1.5 Soporte múltiples pantallas.

- Como sabemos en el mercado existen diferentes dispositivos: teléfonos, tabletas, etc. y ahora ... con las nuevas versiones también relojes, pulseras, televisores, etc.
- Cada uno tiene sus propios tamaños y resoluciones.
- Android proporciona herramientas para adaptarse a toda esa variedad. Por ejemplo, podríamos diseñar una aplicación cuya interfaz gráfica fuera diferente de un teléfono a una tableta, o si el teléfono esta en horizontal o vertical.
- El sistema también intenta escalar la aplicación para que se adapte a diferentes tipos (tamaños de pantalla / resoluciones), pero si usamos píxeles para definir los elementos que componen una pantalla puede suceder lo siguiente:



- Los dispositivos tienen el mismo tamaño de pantalla.
- La resolución del dispositivo varía de menor a mayor (de izquierda a derecha).
- Observar cómo al configurar un objeto para que ocupe  $X * Y$  **píxeles**, estos objetos son más pequeños a medida que la resolución de la pantalla es mayor.
- Tened en cuenta que los tipos de recursos gráficos que se ven afectados son los siguientes:
  - Los creados por el usuario en la aplicación: por ejemplo el botón.
  - Los añadidos por el usuario a la aplicación (dibujos / imágenes): por ejemplo la imagen de Android.
- El objetivo sería que, independientemente de la resolución de la pantalla, se obtuviera el siguiente resultado: se ven objetos del mismo tamaño incluso con diferentes resoluciones.



Vamos a ver antes de nada una serie de conceptos:

### Tamaños de pantalla y densidad (puntos por pulgada: dpi)

- **Tamaño de pantalla:** el tamaño diagonal de la pantalla física del dispositivo.
  - Para simplificar, Android divide las pantallas, hasta la versión 3.2 de Android, en: **pequeñas (small)**, **normales, grandes (large)** y **extras grandes (extra large)**.
- **Densidad de pantalla (Screen density):** puntos por pulgada (**ppp / (dot per inch (dpi))**). Una pantalla de baja resolución tendrá pocos dpi's en comparación con una pantalla normal o de alta resolución.
- Por simplicidad, Android agrupó las densidades:
  - **ldpi:** bajo ( low 120 ppp)
  - **mdpi:** medio (medium, 160 ppp)
  - **hdpi:** alta (hig, 240 ppp)
  - **xhdpi:** extra alto (extra high, 320 ppp)
  - **xxhdpi:** extra extra alto (extra-extra high, 480 ppp)
  - **xxxhdpi:** extra extra extra alto (extra extra-extra-extra high, 640 ppp)

### Orientación y resolución

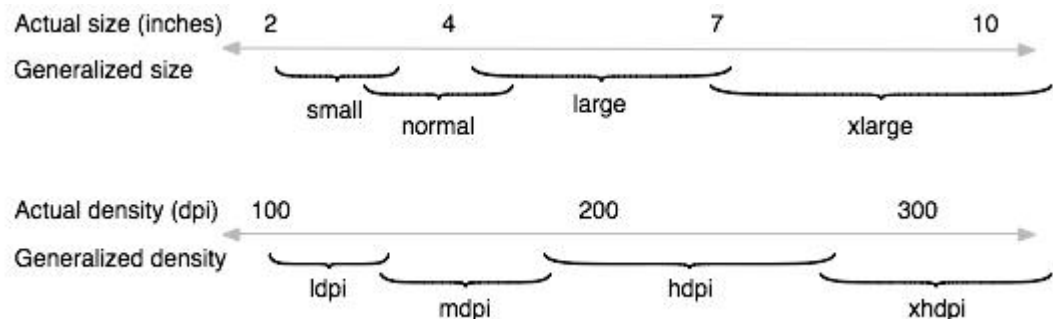
- **Orientación:** Desde el punto de vista del usuario, indica si el dispositivo está en horizontal o vertical. Al girar la pantalla, cambia la relación de altura y ancho.
- **Resolución:** indica el número total de píxeles en la pantalla. Al desarrollar aplicaciones para diferentes pantallas, se debe trabajar con la densidad y el tamaño de la pantalla y no con la resolución total.

### Píxeles independientes de la densidad (dp)

- **Píxel independiente de la densidad (Density-indepent pixel (dp)):** es un píxel virtual que se debe utilizar al definir elementos gráficos en un diseño (layout), tanto para expresar el tamaño de un objeto como su posición.
- Una pantalla mediana (medium) tiene 160 ppp.
- En una pantalla media 1 dp = 1 dpi. Es decir, en una pantalla media, el píxel virtual coincide con el píxel real.
- El tiempo de ejecución es cuando se convierten unidades dp en píxeles físicos (px).
- El factor de conversión es: **px = dp (dpi / 160)**.

- Por ejemplo, 1 dp en una pantalla de densidad:
  - 120 ppp: sería igual a 0,75 px (píxeles físicos).
  - 160 ppp: sería igual a 1 px (píxeles físicos).
  - 240 ppp: equivaldría a 1,5 px (píxeles físicos).
  - 320 ppp: sería igual a 2 px (píxeles físicos).
  - 480 ppp: sería igual a 3 px (píxeles físicos).
  - 640 ppp: sería igual a 4 px (píxeles físicos).
- De esta manera, si siempre se definen las dimensiones y posiciones en dp, siempre realizará una conversión en tiempo de ejecución al número equivalente de píxeles físicos.
- **Scale-independent Pixels (sp)**: esta unidad es igual a dp, pero se utiliza para tamaños de texto. Estos tamaños se pueden ajustar según la densidad de la pantalla y las preferencias del usuario.

La siguiente imagen muestra los rangos entre los que se mueven los distintos tamaños y los distintos tipos de densidad

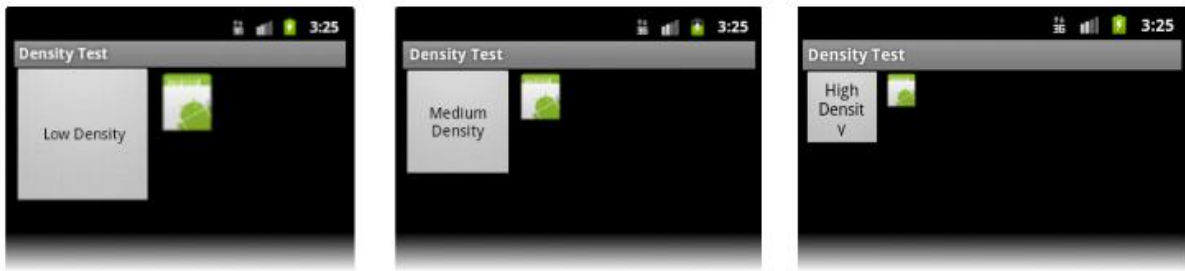


## Las imágenes

Pero aún necesitamos saber qué hacer con las **imágenes que un usuario puede incrustar en una aplicación**.

- Se puede proporcionar un archivo diferente adaptado a las diferentes densidades para cada imagen.
- Si no se proporcionan archivos para las diferentes densidades, el sistema escalará el archivo proporcionado para la densidad media (mdpi) y lo adaptará a la densidad de la pantalla. Pero siempre será mejor proporcionar diferentes versiones de la imagen.
- Para proporcionar versiones de diferente densidad de una imagen debemos seguir la regla: ldpi:mdpi:hdpi:xhdpi:xxhdpi = 3:4:6:8:12 . Es decir, si tomamos como referencia una imagen de densidad media (mdpi) de tamaño X \* Y píxeles, debemos crear nuevas imágenes que mantengan la siguiente relación en tamaño (número de píxeles): ldpi (x0,75), mdpi (x1), hdpi (x1,5) y xhdpi (x2).
- Por ejemplo, si tenemos una imagen de 48x48 píxeles (tamaño del icono de inicio de una aplicación) para una densidad mdpi, deberíamos crear versiones con los siguientes tamaños en píxeles:
  - ldpi: 36 x 36
  - mdpi: 48 x 48
  - hdpi: 72 x 72
  - xhdpi: 96 x 96

Una vez visto todo lo anterior, volvamos a ver de nuevo las imágenes donde se usan los píxeles para los tamaños y las posiciones:



y la que usa **dp** y diferentes versiones de una misma imagen:



observad como coinciden ambos en el caso de la densidad media.

Referencias:

- [http://developer.android.com/guide/practices/screens\\_support.html](http://developer.android.com/guide/practices/screens_support.html)
- <http://developer.android.com/design/style/iconography.html>
- Para las dimensiones: <http://developer.android.com/guide/topics/resources/more-resources.html#Dimension>