

**MP_0489. Programación
multimedia y dispositivos móviles
UF2. Programación de
aplicaciones para dispositivos móviles**

2.1. Gráficos y estilos

Índice

☰	Objetivos	3
☰	Drawables	4
☰	Tipos de elementos Drawables	7
☰	Images	24
☰	Styles	30
☰	Themes	33
☰	Resumen	36

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

1

Conocer los distintos elementos *Drawable*.

2

Crear archivos *9-patch* para trabajar con distintas resoluciones.

3

Aprender cómo trabajar con imágenes y vectores.

4

Descubrir *Style* y *Themes* para asignar estilos a toda la aplicación.

¡Ánimo y adelante!

Drawables

Los elementos *drawables* son imágenes compiladas que se pueden usar en una aplicación.

Android proporciona clases y recursos para **incluir imágenes en una aplicación, con un impacto mínimo en su rendimiento.**



Un *drawable* es un gráfico que se puede dibujar en la pantalla.

Podemos recuperar *drawables* utilizando la API con *getDrawable()*. También podemos aplicar un *drawable* a un recurso XML, utilizando atributos como *android:drawable* y *android:icon*.

Cómo usar *drawables*

A la hora de mostrar un *drawable* usamos la clase *ImageView* para crear una *View*.

En el elemento *ImageView* del archivo XML, definimos cómo se muestra el *drawable* y dónde se ubica. Por ejemplo, este *ImageView* muestra una imagen llamada "elefante.png":

```
<ImageView  
    android:id="@+id/tiles"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/elefante" />
```

Atributos de *ImageView*

El atributo *android:id* establece un nombre de acceso directo, que usaremos para llamar a la imagen más adelante.

Los atributos *android:layout_width* y *android:layout_height* especifican el tamaño de la *View*.

En el ejemplo anterior, la altura y el ancho se establecen en *wrap_content*, lo que significa que la *View* es lo suficientemente grande como para encerrar la imagen dentro de ella, junto con el *padding*.

El atributo *android:src* proporciona la ubicación en la que se almacena esta imagen. Si tenemos versiones apropiadas para diferentes resoluciones de pantalla, las pondremos en carpetas llamadas *res/drawable-[density]*.

Por ejemplo, guardamos una versión de elefante.png apropiada para pantallas hdpi en *res/drawable-hdpi/elefante.png*.

ImageView también tiene atributos que podemos usar para **recortar una imagen**, si es demasiado grande o tiene una relación de aspecto diferente a la del diseño o la *View*.



Para representar un *drawable* en su aplicación, usamos la clase *Drawable* o una de sus subclases. Por ejemplo, este código recupera la imagen `elefante.png` como *drawable*.

```
Resources res = getResources();
Drawable drawable = res.getDrawable(R.drawable.elefante);
```

Tipos de elementos *drawables*

Android incluye **varios tipos de elementos *drawables***, como por ejemplo:

- Archivos de imagen.
- Archivos *9-patch*.
- *Layer list*.
- *Shape list*.
- *State list*.
- *Level list*.
- *Transition drawable*.
- Imágenes vectoriales.

Tipos de elementos Drawables

Sigamos analizando los diferentes elementos *drawables* que tenemos a nuestra disposición.

Y lo haremos analizando uno por uno los **distintos tipos** que podemos utilizar.

Archivos de imagen



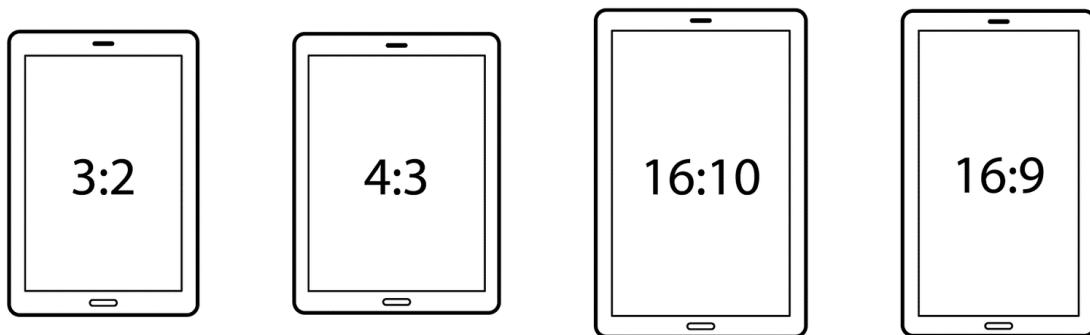
Un archivo de imagen es un **archivo de mapa de bits genérico**.

Android admite archivos de imagen en varios formatos:

- WebP (preferido).
- PNG (preferido).
- JPG (aceptable).
- Los formatos GIF y BMP son compatibles, pero no se recomiendan.

El formato WebP es totalmente compatible con Android 4.2. WebP se comprime mejor que otros formatos para obtener una compresión sin pérdidas, lo que podría generar imágenes un 25% más pequeñas que los formatos JPEG. Se puede convertir las imágenes PNG y JPEG existentes en formato WebP antes de cargarlas.

Guardamos los archivos de imagen en la carpeta **res/drawable** y los usamos con el atributo **android:src**, para un *ImageView* o para crear una clase de *BitmapDrawable*.



Relación de aspecto en diferentes tamaños de pantalla.

Las imágenes se ven diferentes en pantallas con diferentes densidades de píxeles y relaciones de aspecto, por lo que es importante **usar siempre las del tamaño adecuado**, ya que pueden consumir una gran cantidad de espacio en el disco y afectar al rendimiento de la aplicación.

Archivos 9-patch



Una imagen 9-patch es una **imagen PNG en la que se definen regiones extensibles**.

Usamos una imagen 9-patch como imagen de fondo con el fin de **asegurarnos de que la View se vea correctamente para diferentes tamaños de pantalla y orientaciones**.

Por ejemplo, en una *View* que tiene *layout_width* establecido en "wrap_content", la *View* se mantiene lo suficientemente grande como para incluir su contenido (más el *padding*).

Si usamos una imagen PNG normal como imagen de fondo para la *View*, en algunos dispositivos puede verse demasiado pequeña, ya que la *View* se extiende para acomodar su contenido. Si en su lugar usamos una imagen *9-patch*, esta se estira a medida que la *View* se estira.



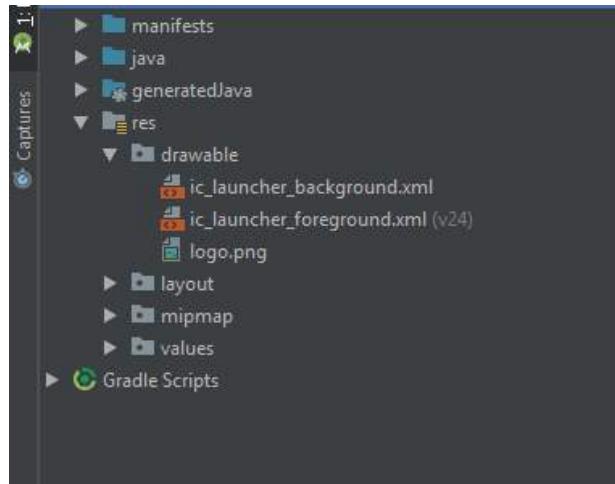
El *widget* de *Button* de Android es un ejemplo de una *View* que usa un *9-patch* como imagen de fondo. El *9-patch* se estira para acomodar el texto o la imagen dentro del botón.

Los archivos *9-patch* se guardan con una extensión **.9.png**. Los usamos con el atributo *android:src* para un *ImageView*, o para crear una clase de *NinePatchDrawable*.

Para **crear una imagen 9-patch** podemos usar la herramienta *Draw 9-patch* en Android Studio. Esta herramienta nos permite comenzar con un PNG normal y definir un borde de píxel alrededor de la imagen en lugares donde está bien que el sistema Android extienda la imagen, si es necesario.

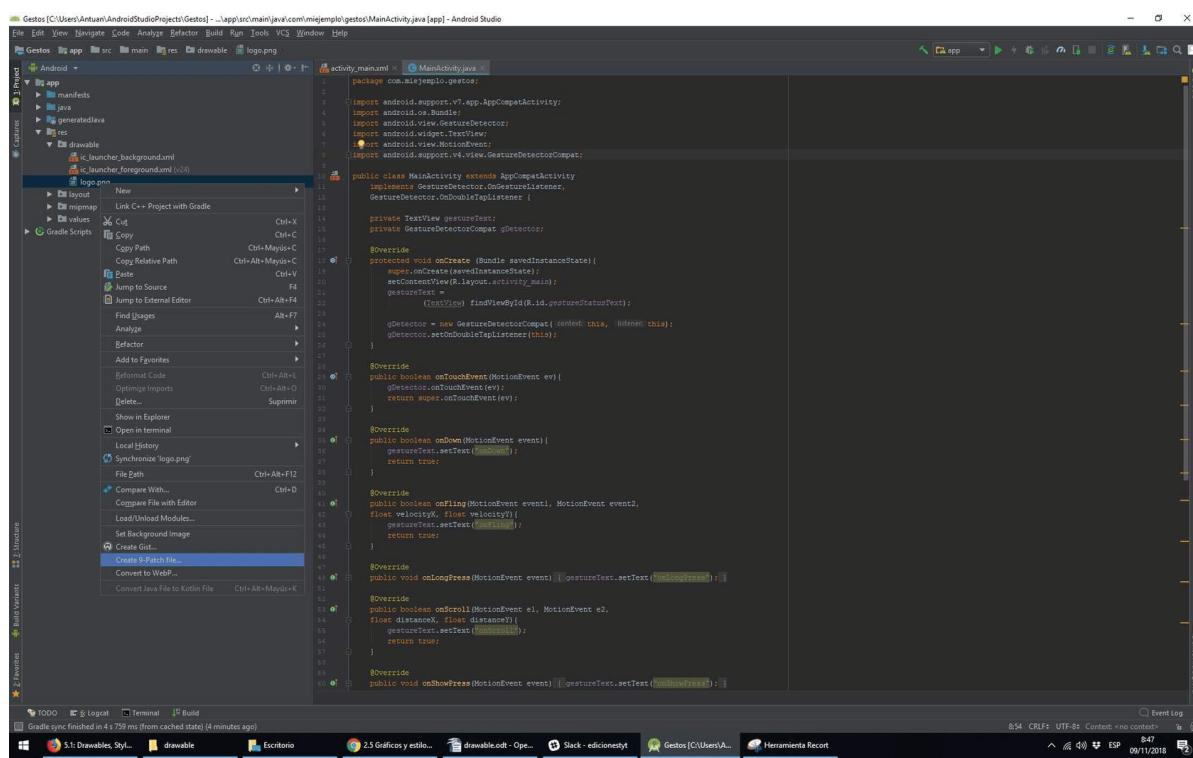
Veamos cómo crear un archivo 9-patch.

Paso 1



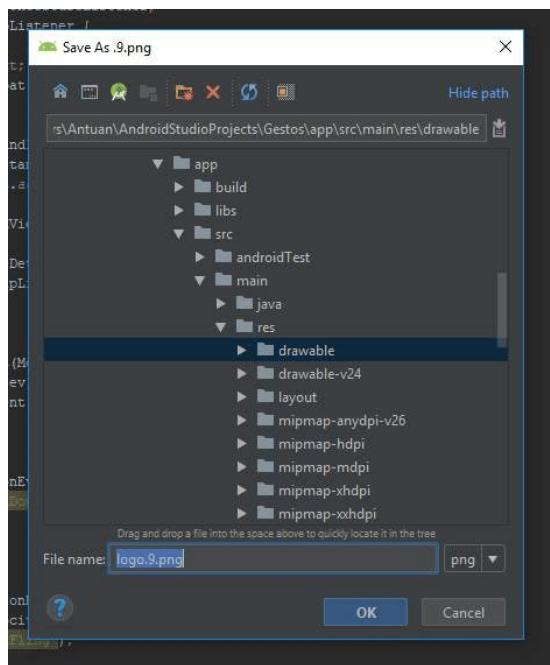
Ponemos un archivo PNG en la carpeta res/drawable.

Paso 2



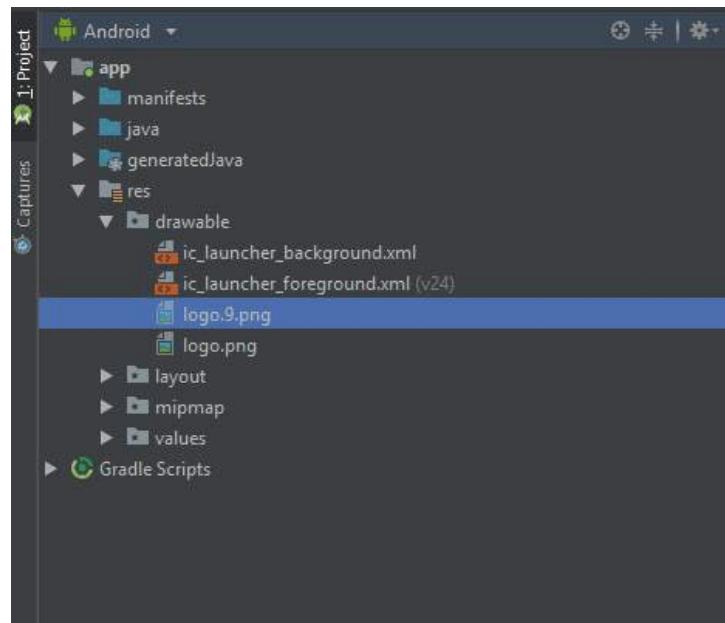
Hacemos clic derecho en el archivo y elegimos *Create 9-patch file...*

Paso 3



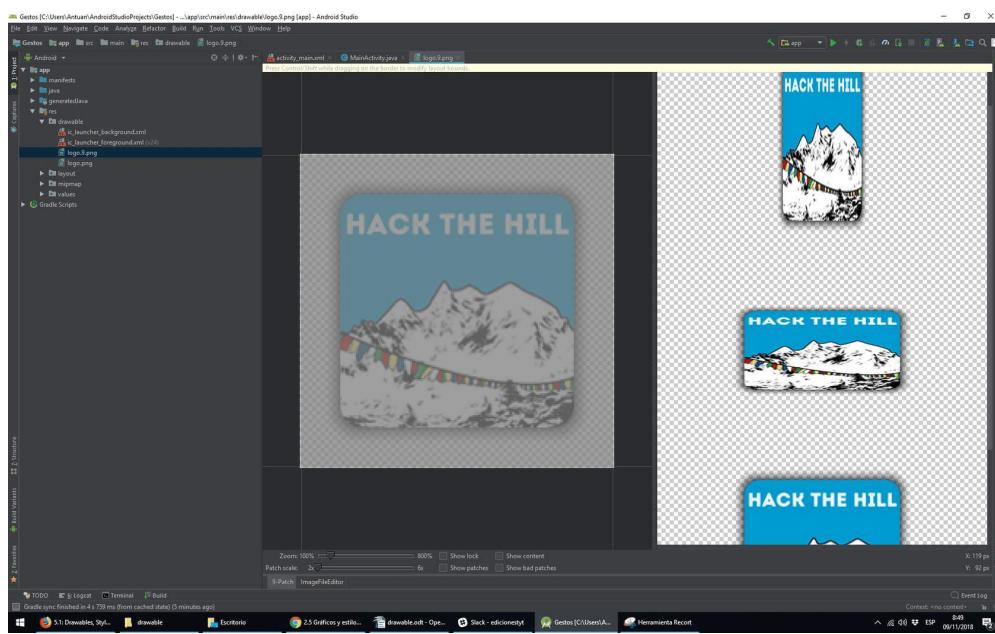
Android Studio nos preguntará dónde queremos guardarlo.

Paso 4



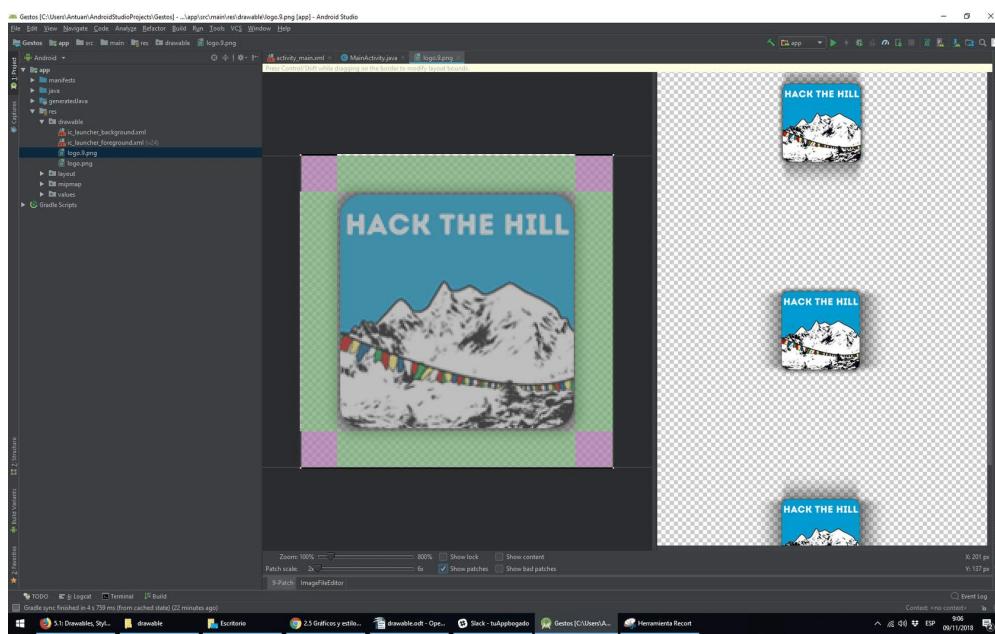
Android Studio guardará el archivo con una extensión **.9.png**.

Paso 5



Hacemos doble clic en el archivo .9.png para abrirlo en el editor. A la derecha vemos cómo se mostrará.

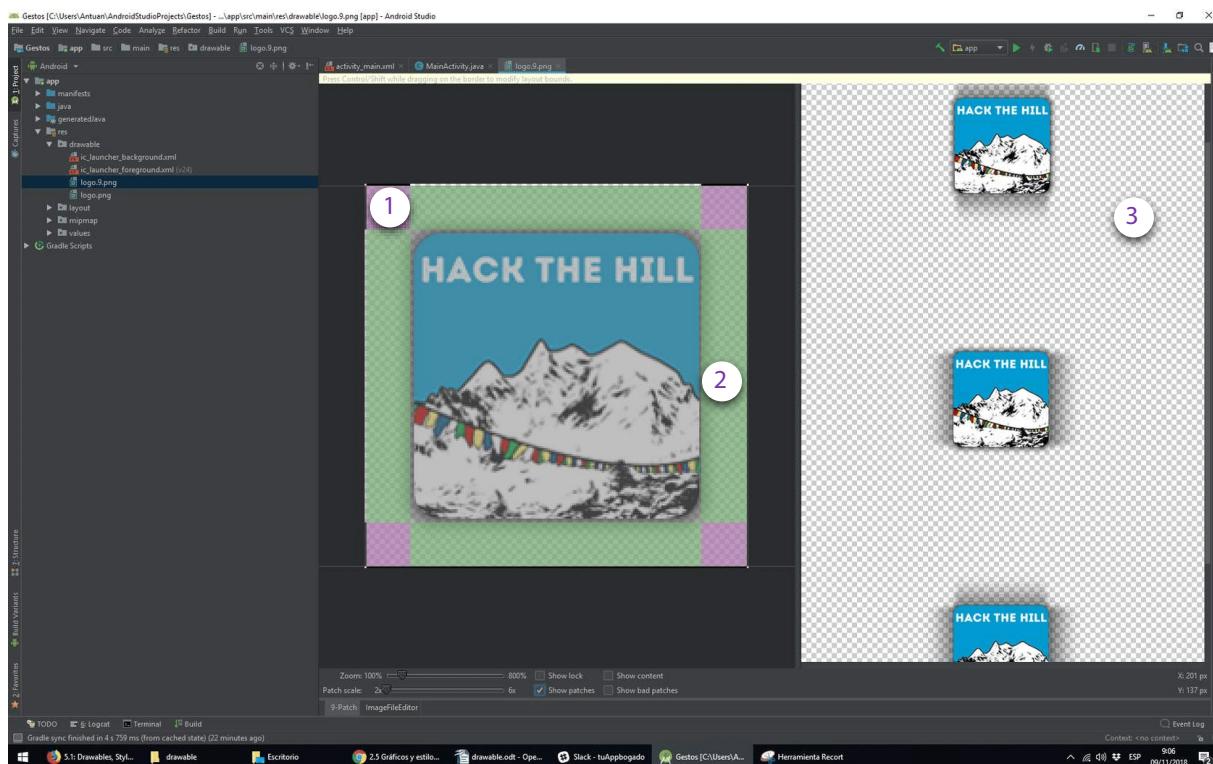
Paso 6



Especificamos qué regiones de la imagen se pueden estirar (en verde).

Ya tenemos nuestro archivo *9-patch* preparado.

Así se vería la imagen:



1. Zona que no se estirará.
2. Zona que se estirará.
3. Vista previa de cómo se verá la imagen.

Layer List

En Android podemos construir una imagen al juntar otras imágenes, al igual que en Gimp y otros programas de manipulación de imágenes. Cada capa está representada por un ***drawable*** individual.



Los elementos ***drawables*** que forman una sola imagen se organizan y administran en un elemento ***layer list***, en formato XML. Dentro de *layer list*, cada *drawable* está representado por un ítem.

Las capas se dibujan una sobre otra en el orden definido en el archivo XML, lo que significa que **el último *drawable* de la lista se dibuja en la parte superior**.

Por ejemplo, esta lista de capas *drawable* se compone de tres *drawables* superpuestos uno sobre otro:

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <bitmap android:src="@drawable/android_red"
            android:gravity="center" />
    </item>
    <item android:top="10dp" android:left="10dp">
        <bitmap android:src="@drawable/android_green"
            android:gravity="center" />
    </item>
    <item android:top="20dp" android:left="20dp">
        <bitmap android:src="@drawable/android_blue"
            android:gravity="center" />
    </item>
</layer-list>
```



Layer Drawable.

En el XML que define esta lista de capas, la imagen de *android_blue* se define en último lugar, por lo que se dibuja en último lugar y se muestra en la parte superior.

Shape list



Una *Shape list drawable* es un rectángulo, un óvalo, una línea o un anillo que se define en XML. Podemos especificar el tamaño y el estilo de la forma usando atributos XML.

Por ejemplo, en este archivo XML crearemos un rectángulo con esquinas redondeadas y un degradado de color. El color de *padding* del rectángulo cambia de blanco (#000000), en la esquina inferior izquierda, a azul (#000odd) en la esquina superior derecha. El atributo *angle* determina cómo se inclina el degradado:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="8dp" />
    <gradient
        android:startColor="#000000"
        android:endColor="#0000dd"
        android:angle="45"/>
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
</shape>
```

Si el archivo XML de formas *drawable* se guarda en res/drawable/gradient_box.xml, el siguiente *layout* XML aplica la forma *drawable* como fondo de una *View*.

```
<TextView  
    android:background="@drawable/gradient_box"  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content" />
```



here is a color
gradient...

En el siguiente código te mostramos **cómo obtener la forma *drawable* mediante programación y utilizarla como fondo para una *View***, como alternativa a la definición del atributo de fondo en XML:

```
Resources res = getResources();  
Drawable shape = res.getDrawable(R.drawable.gradient_box);  
  
TextView tv = (TextView) findViewById(R.id.textview);  
tv.setBackground(shape);
```

Podemos establecer otros atributos para una forma *drawable*. La sintaxis completa es esta:

```

<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=["rectangle" | "oval" | "line" | "ring"] >
    <!-- If it's a line, the stroke element is required. -->
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />
    <gradient
        android:angle="integer"
        <!-- The angle must be 0 or a multiple of 45 -->
        android:centerX="float"
        android:centerY="float"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
        android:startColor="color"
        android:type=["linear" | "radial" | "sweep"]
        android:useLevel=["true" | "false"] />
    <padding
        android:left="integer"
        android:top="integer"
        android:right="integer"
        android:bottom="integer" />
    <size
        android:width="integer"
        android:height="integer" />
    <solid
        android:color="color" />
    <stroke
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
</shape>
```

State list

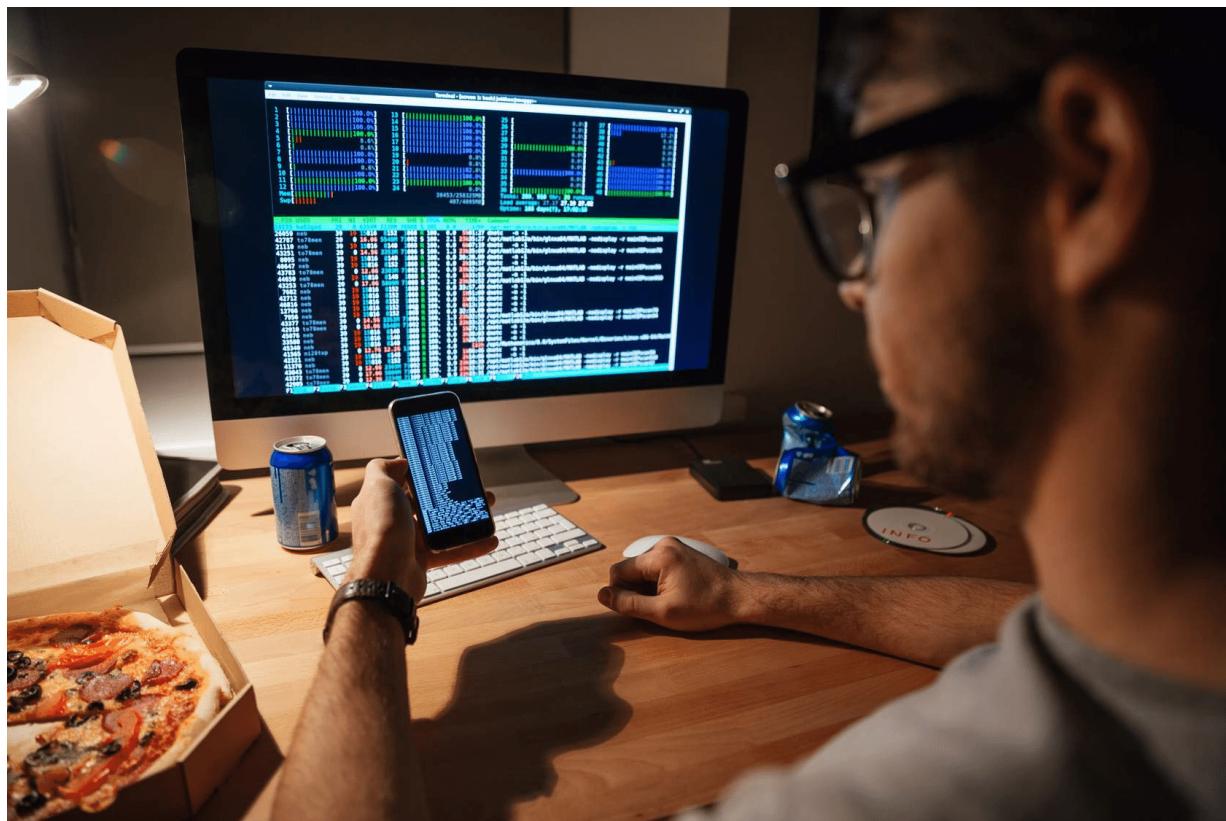


Un objeto *StateListDrawable* es un objeto que usa una imagen diferente para representar el mismo objeto, dependiendo del estado en que se encuentre dicho objeto.

Por ejemplo, un *widget* de botón puede existir en uno de varios estados (*pulsado*, *enfocado*, *desplazado* o no). Usando una *StateListDrawable* podemos proporcionar una imagen de fondo diferente para cada estado.

Podemos describir los distintos estados en un archivo XML. Cada gráfico está representado por un ítem dentro de un único elemento selector. Cada ítem usa un **atributo state_** para indicar la situación en la que se usa el gráfico.

Durante cada cambio de estado, Android recorre la lista de estados de arriba a abajo. Se utiliza el primer elemento que coincide con el estado actual, lo que significa que la selección no se basa en la "mejor coincidencia", sino que **se elige simplemente el primer elemento que cumple los criterios mínimos del estado**.



La lista de estados, en el siguiente ejemplo, define qué imagen se muestra para un botón cuando el botón está en diferentes estados.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!-- pulsado-->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!-- enfocado-->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!-- sobre-->
    <item android:drawable="@drawable/button_normal" /> <!-- normal-->
</selector>
```

Otros estados posibles son:

- *android:state_selected*.
- *android:state_checkable*.
- *android:state_checked*.

Level List



Una *Level list drawable* define **drawables** alternativos, asignando a cada uno un valor numérico máximo. Para seleccionar qué *drawable* utilizar, llamamos al método *setLevel()*.

Por ejemplo, el siguiente XML define una *Level list drawable* que incluye dos elementos alternativos, *status_off* y *status_on*:

```
<?xml version="1.0" encoding="utf-8"?>
<level-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@drawable/status_off"
        android:maxLevel="0" />
    <item
        android:drawable="@drawable/status_on"
        android:maxLevel="1" />
</level-list>
```

- Para seleccionar `status_off`, llamamos a `setLevel(0)`.
- Para seleccionar `status_on`, llamamos a `setLevel(1)`.



Un **ejemplo de uso de un `LevelListDrawable`** es un icono indicador de nivel de batería, que usa diferentes imágenes para indicar diferentes niveles de batería actuales.

Transition Drawable



Un **`TransitionDrawable`** es un **`drawable`** que se desvanece entre otros dos **`drawables`**.

Para definir un *Transition drawable* en XML usamos el elemento *transition*. Cada *drawable* está representado por un ítem dentro del elemento *transition*, y no podemos usar más de dos ítems.

En este ejemplo, el *drawable* se desvanece entre un estado "encendido" y un estado "apagado":

```
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/on" />
    <item android:drawable="@drawable/off" />
</transition>
```

- Para hacer una transición hacia adelante, es decir, cambiar del primer *drawable* al segundo, llamamos a *startTransition()*.
- Para hacer la transición al revés, lo haremos con *reverseTransition()*.

Cada uno de estos métodos toma un argumento de tipo *int*, que representa el **número de milisegundos para la transición**.

Imágenes vectoriales



Los vectores *drawables* son imágenes definidas por una ruta, que podemos definir a partir de Android 5.0 (nivel de API 21).

Los vectores son **escalables sin perder definición**. La mayoría de los vectores *drawables* utilizan archivos SVG, que son archivos de texto plano, o archivos binarios

comprimidos, que incluyen coordenadas bidimensionales para dibujar la imagen en la pantalla.

Debido a que los archivos SVG son texto, ocupan menos espacio que la mayoría de archivos de imagen. Además, solo necesitan un archivo para cada densidad de pantalla, no como en el caso de las imágenes de mapas de bits.

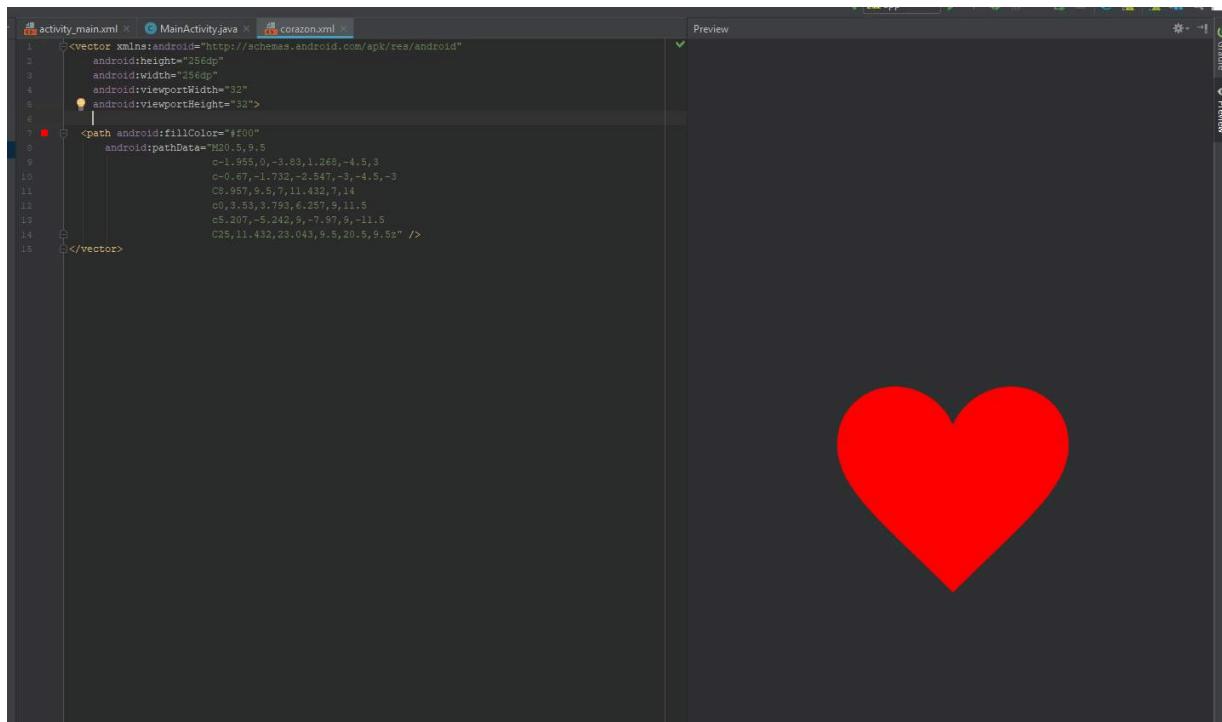
Para traer una imagen vectorial existente o un ícono de diseño de materiales a nuestro proyecto de Android Studio como un *drawable vectorial*, debemos implementar el siguiente código:

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"  
        android:height="256dp"  
        android:width="256dp"  
        android:viewportWidth="32"  
        android:viewportHeight="32">  
  
    <path android:fillColor="#f00"  
          android:pathData="M20.5,9.5  
                      c-1.955,0,-3.83,1.268,-4.5,3  
                      c-0.67,-1.732,-2.547,-3,-4.5,-3  
                      C8.957,9.5,7,11.432,7,14  
                      c0,3.53,3.793,6.257,9,11.5  
                      c5.207,-5.242,9,-7.97,9,-11.5  
                      C25,11.432,23.043,9.5,20.5,9.5z" />  
  
</vector>
```



corazon.xml
708 B





Si ya tenemos una imagen en formato SVG, podemos obtener la información de `pathData` en Android Studio, haciendo clic derecho en la carpeta `drawable` y **seleccionando New Vector Asset para abrir la herramienta Vector Asset Studio**. Usaremos esta herramienta para importar un archivo SVG local.

Recuerda que las imágenes vectoriales se representan en Android como **objetos VectorDrawable**.

Images

Las imágenes en Android, desde los iconos *launcher* hasta los *banners*, **se pueden utilizar de muchas maneras.**

Crear iconos

Cada aplicación requiere de, al menos, un ícono. Además, a menudo se incluyen íconos de la *Action bar*, notificaciones y otros usos.

Hay dos **posibilidades para crear íconos:**

1. Crear un conjunto de archivos de imagen del mismo ícono en diferentes resoluciones y tamaños, para que el ícono se vea igual en todos los dispositivos aunque tengan diferentes densidades de pantalla. Para ello podemos usar **Image Asset Studio**.
2. Usar **drawables vectoriales**, que se escalan automáticamente sin que la imagen se vuelva pixelada o borrosa. Para ello podemos usar **Vector Asset Studio**.

1

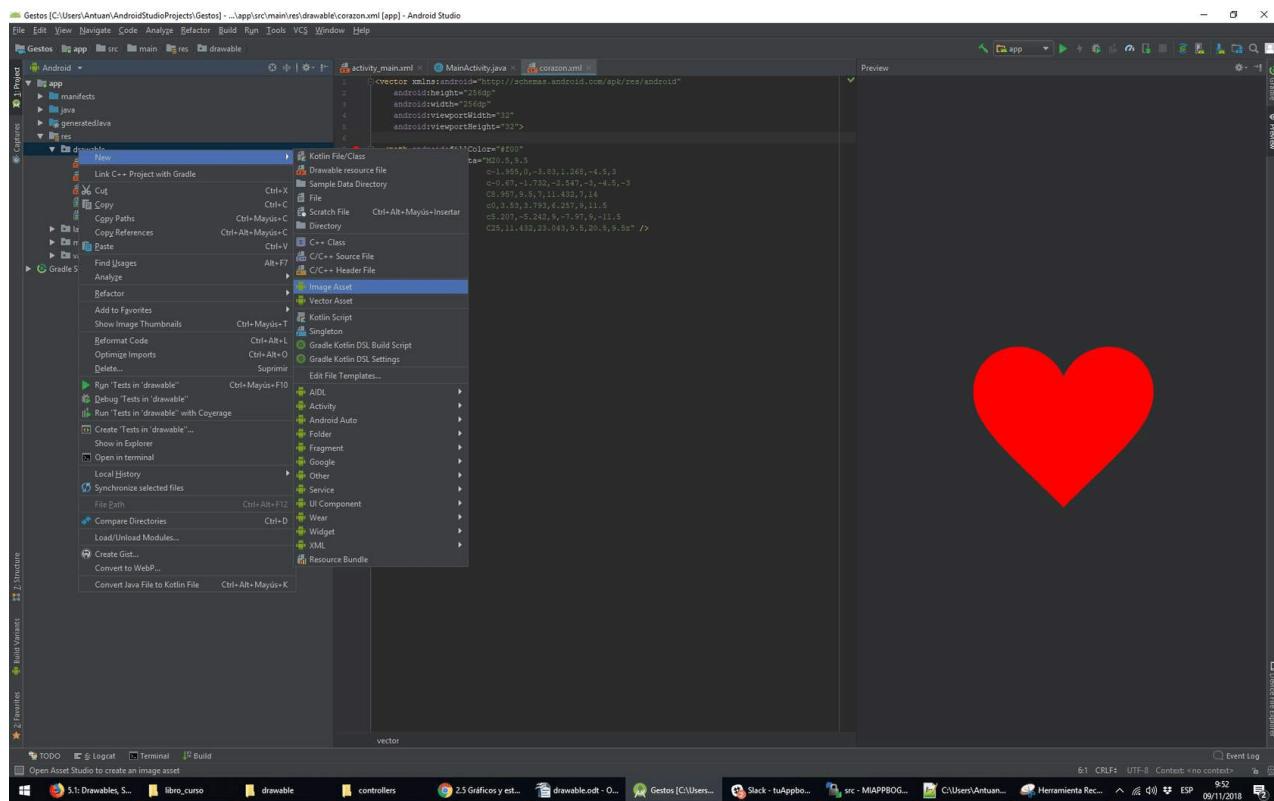
Image Asset Studio.

Android Studio incluye una herramienta llamada Image Asset Studio que **nos ayuda a generar nuestros propios íconos de aplicaciones a partir de íconos de Material Design, imágenes personalizadas y cadenas de texto.**

Esta herramienta genera un conjunto de iconos con la resolución adecuada para cada densidad de pantalla generalizada que admite la aplicación. Image Asset Studio coloca los iconos recién generados en carpetas específicas de densidad debajo de la carpeta res/ en el proyecto. Android usa el recurso apropiado, en función de la densidad de pantalla del dispositivo en el que se está ejecutando la aplicación.

Image Asset Studio nos ayuda a generar los siguientes tipos de iconos:

- Iconos *launcher*.
- *Action bar* e iconos de pestañas.
- Iconos de notificación.



Así abrimos Image Asset Studio en el menú de Android Studio.

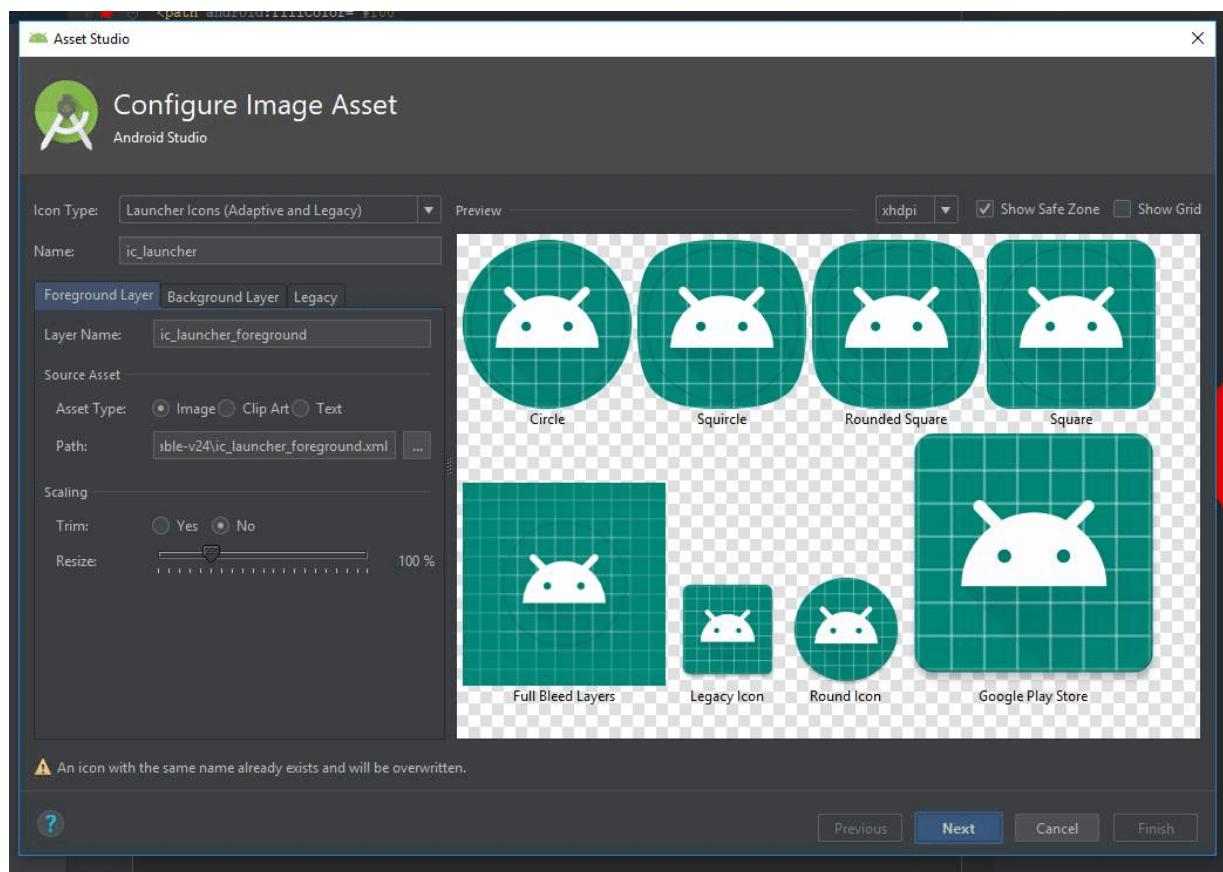


Image Asset Studio.

2

Vector Asset Studio.

A partir de API 21, se pueden usar vectores *drawables* en lugar de archivos de imagen para crear iconos. Veamos las **ventajas y desventajas de su uso**:

VENTAJAS

- Los vectores *drawables* **pueden reducir el tamaño del archivo APK**, porque no tienen que incluir varias versiones de cada imagen de icono.
- Podemos usar una imagen vectorial para **escalar sin problemas a cualquier resolución**.
- Es probable que los usuarios prefieran descargar una aplicación más pequeña.

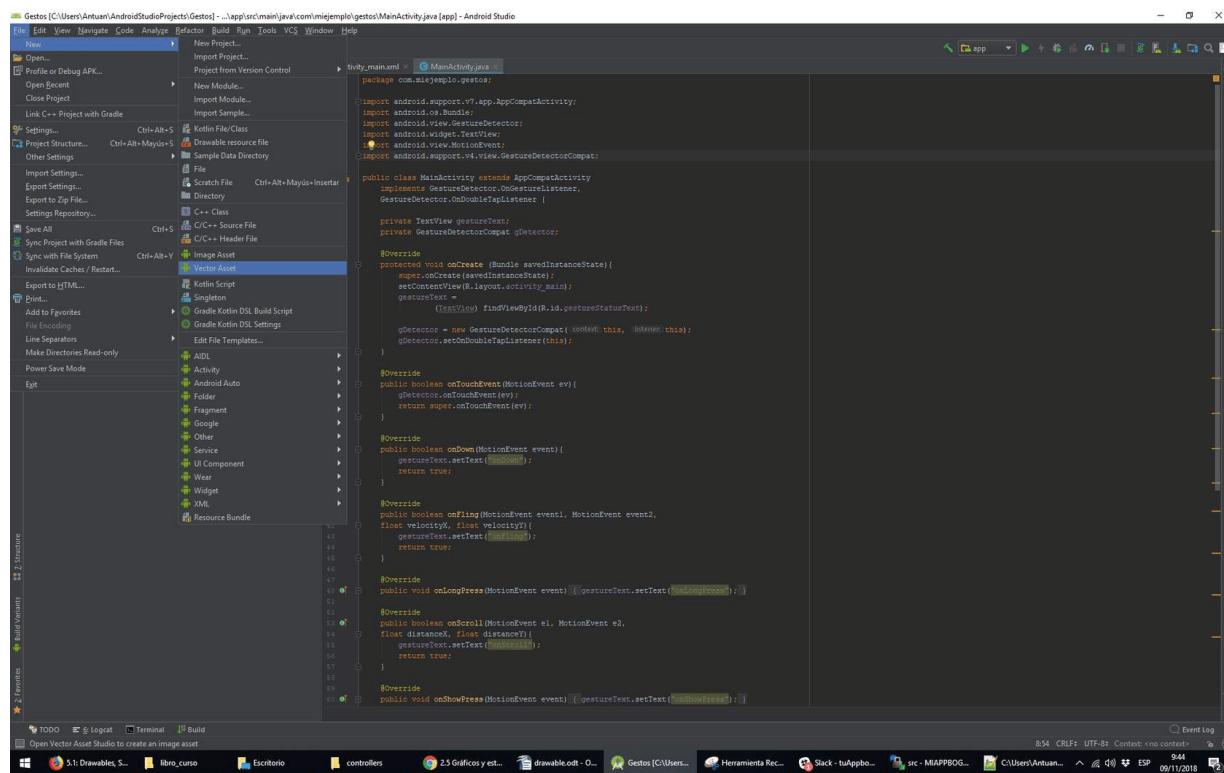
DESVENTAJAS

- Los *drawable* vectoriales **pueden incluir solo una cantidad limitada de detalles**. Se utilizan principalmente para iconos menos detallados, como los de *Material Design*. Los iconos con más detalle suelen necesitar archivos de imagen.
- Los *drawables* vectoriales **no son compatibles con dispositivos que ejecutan API nivel 20 o inferior**.

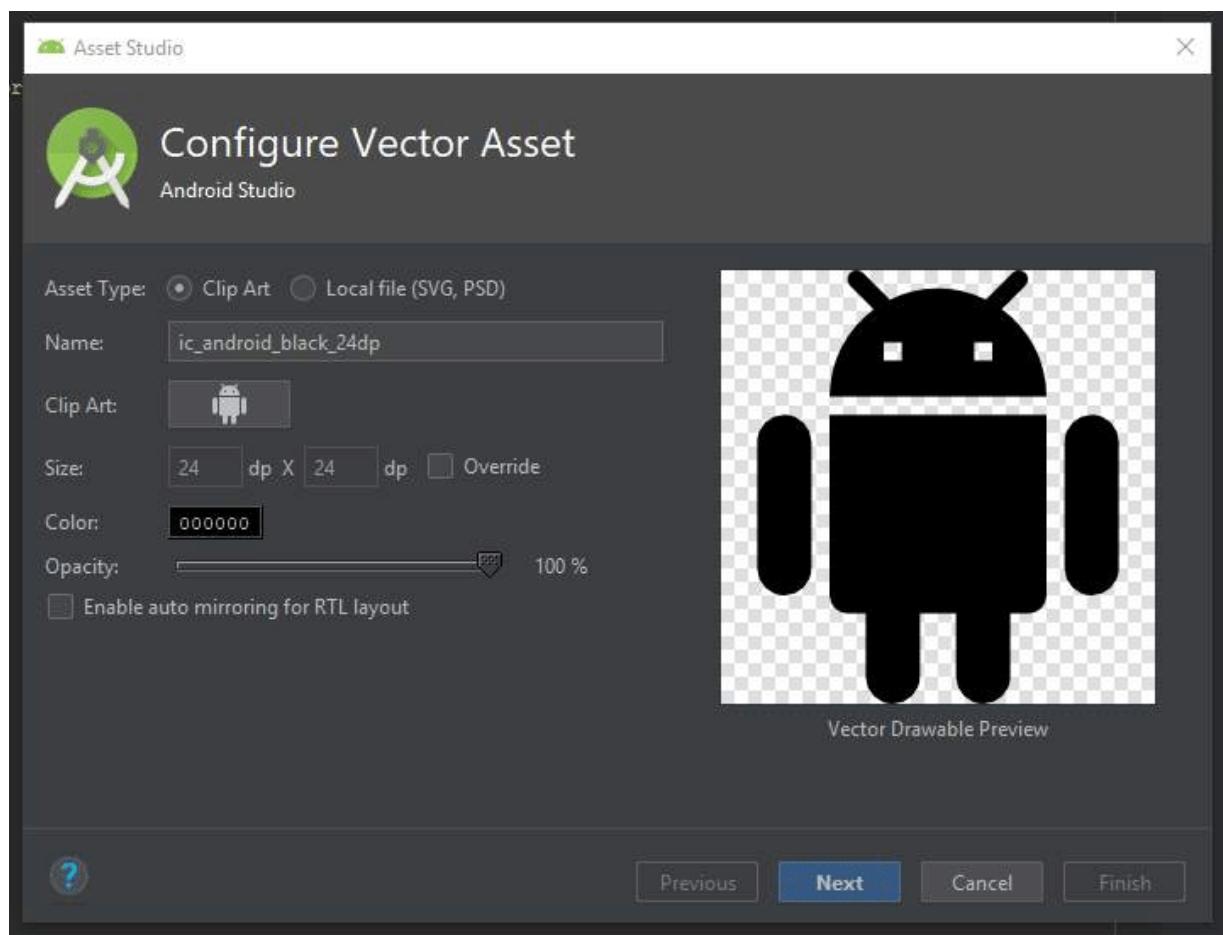
Para usar vectores *drawables* en dispositivos con nivel de API 20 o inferior, debemos elegir entre dos métodos de compatibilidad con versiones anteriores:

1. De forma predeterminada, el sistema crea versiones de mapa de bits de sus vectores *drawables* en diferentes resoluciones en el momento de la compilación. Esto permite que los iconos se ejecuten en dispositivos que no pueden dibujar vectores.
2. La clase *VectorDrawableCompat* en la biblioteca de soporte de Android permite admitir vectores *drawables* en Android 2.1 (nivel API 7) y superior.

Vector Asset Studio es una herramienta que nos ayuda a agregar iconos de *Material Design* y *drawables* vectoriales al proyecto de Android.



Así abrimos Vector Asset Studio en el menú de Android Studio.



Vector Asset Studio.

Otros tipos de imagen

Ya hemos visto qué sucede con los iconos. Pero también podemos encontrar los *banners*, las imágenes de perfil de usuario y otros tipos de imágenes en **distintas formas y tamaños**.

En muchos casos, son más grandes de lo que deberían para una interfaz de usuario (UI) típica de la aplicación. Por ejemplo, la aplicación de *Galería del sistema* muestra las fotos tomadas con la cámara de un dispositivo Android, fotos que suelen tener una resolución mucho más alta que la densidad de la pantalla del dispositivo.



Una imagen con mayor resolución no proporciona ningún beneficio, pero ocupa más memoria.

Los dispositivos Android tienen **memoria finita**, por lo que lo ideal sería **cargar solo una versión de menor resolución de una foto en la memoria**. La versión de menor resolución debe coincidir con el tamaño del componente UI que lo muestra. Una imagen con una resolución más alta no proporciona ningún beneficio visible, pero ocupa una valiosa memoria y agrega una sobrecarga de rendimiento adicional, debido a la escala adicional sobre la marcha.

Podemos **modificar esas imágenes manualmente** antes de utilizarlas en la aplicación, o bien **buscar alguna librería externa** que haga esa labor por nosotros.

Styles

Un style es una colección de atributos que definen el aspecto y el formato de una View.

El uso de estilos permite **mantener estos atributos comunes en una ubicación y aplicarlos a cada TextView, usando una sola línea de código en XML.**

Se puede aplicar el mismo *style* a cualquier número de *Views* en la aplicación; por ejemplo, varias *Views* de un texto pueden tener el mismo tamaño y diseño.

Podemos definir estilos por nosotros mismos, o bien usar uno de los estilos de plataforma que ofrece Android.

Definir y aplicar estilos

Para crear un estilo, **agregamos un elemento *style* dentro de un elemento *resources* en cualquier archivo XML ubicado en la carpeta res/values/**. Cuando creamos un proyecto en Android Studio, se crea un archivo res/values/styles.xml.

Un elemento **style** incluye:

1

Un atributo de **name**. Usamos el nombre del **style** cuando lo aplicamos a una **View**.

2

Un atributo **parent** (opcional). Lo veremos con más detalle en el siguiente punto.

3

Un número de ítems como elementos secundarios de **style**. Cada ítem incluye un atributo **style**.

En este ejemplo crearemos un **style** que formatee el texto para usar un tipo de letra *monospace* gris claro:

```
<resources>
    <style name="CodeFont">
        <item name="android:typeface">monospace</item>
        <item name="android:textColor">#D7D6D7</item>
    </style>
</resources>
```

El siguiente XML aplica el nuevo **style** *CodeFont* a una **View**:

```
<TextView
    style="@style/CodeFont"
    android:text="@string/code_string" />
```

Herencia



Un nuevo **style** puede heredar las propiedades de un **style** existente.

Cuando creamos un *style* que hereda propiedades, definimos solo las propiedades que deseamos cambiar o agregar. El nuevo *style* puede heredar propiedades de los *styles* de plataforma y de los que hayamos creado. Para heredar un *style* de la plataforma, usamos el atributo *parent* para especificar el ID de recurso de *style* que queramos heredar.

Por ejemplo, aquí heredamos la apariencia de texto predeterminada de la plataforma Android (el estilo de apariencia de texto) y cambiamos su color:

```
<style name="GreenText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
</style>
```

Para aplicar este *style*, usamos `@style/GreenText`.

Para heredar un *style* que nosotros mismos creamos, usamos el nombre del *style* que se desea heredar como primera parte del nombre del nuevo *style* y separamos las partes con un punto:

```
name="StyleToInherit.Qualifier"
```

Para crear un *style* que herede el estilo *CodeFont* definido anteriormente, usamos *CodeFont* como primera parte del nombre:

```
<style name="CodeFont.RedLarge">
    <item name="android:textColor">#FF0000</item>
    <item name="android:textSize">34sp</item>
</style>
```

Este ejemplo incluye el atributo de *typeface* del *style* original *CodeFont*, reemplaza el atributo *textColor* original con rojo y agrega un nuevo atributo, *textSize*. Para aplicar este *style* usamos `@style/CodeFont.RedLarge`.

Themes

Los *themes* se crean de la misma manera que los *styles*.

Es decir, **agregando un elemento *style* dentro de un elemento *resources*** en cualquier archivo XML ubicado en la carpeta res/values/.

Diferencias entre un *style* y un *theme*

- 1 Un estilo se aplica a una *View*. En XML, aplicamos un estilo utilizando el **atributo *style***.
- 2 Sin embargo, un *theme* se aplica a toda una actividad o aplicación, en lugar de a una *View* individual. En XML, aplicamos un *theme* utilizando el atributo ***android:theme***.
- 3 Cualquier estilo puede ser usado como *theme*. Por ejemplo, podríamos aplicar el estilo *CodeFont* como *theme* para una actividad, y todo el texto dentro de la actividad usaría una fuente de *CodeFont* gris.

Aplicar *themes*

Para **aplicar un *theme* a la aplicación**, lo declaramos en un elemento *application* dentro del archivo *AndroidManifest.xml*. Este ejemplo aplica el tema *AppTheme* a toda la aplicación:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.exampledomain.myapp">
    <application>
        ...
        android:theme="@style/AppTheme">
    </application>
    ...

```

Para aplicar un **theme** a una actividad, lo declaramos en un elemento **activity** dentro del archivo **AndroidManifest.xml**. En este ejemplo, el atributo *android:theme* aplica el *theme* de la plataforma **Theme_Dialog** a la actividad:

```
activity android:theme="@android:style/Theme.Dialog"
```

Theme predeterminado

Cuando creamos un nuevo proyecto en Android Studio, se define un **theme** predeterminado dentro del archivo **styles.xml**. Por ejemplo, este código podría estar en el archivo **styles.xml**:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

En el ejemplo, *AppTheme* hereda de *Theme.AppCompat.Light.DarkActionBar*, que es uno de los muchos *themes* que la plataforma Android tiene disponibles.

Styles de plataforma y themes

La plataforma Android proporciona una colección de *styles* y *themes* que podemos usar en una aplicación. Para ver una lista de todos ellos, podemos buscar en dos sitios:

- La clase *R.style* enumera la mayoría de los *styles* y *themes* de plataforma disponibles.
- La clase *support.v7.appcompat.R.style* enumera algunos más. Estos *styles* y *themes* tienen "AppCompat" en sus nombres, y son compatibles con la biblioteca de aplicaciones v7.



Los nombres de *style* y *theme* incluyen **guiones bajos**. Para usarlos en el código, **reemplazamos los guiones bajos con puntos**.

Observa el ejemplo: **cómo aplicar el theme *Theme_NoTitleBar* a una actividad.**

```
<activity android:theme="@android:style/Theme.NoTitleBar"
```

Otro ejemplo: **cómo aplicar el style *AlertDialog_AppCompat* a una View.**

```
<TextView  
    style="@style/AlertDialog.AppCompat"  
    android:text="@string/code_string" />
```

Resumen

Hemos terminado la lección, repasemos los puntos más importantes que hemos tratado.

- A lo largo de esta unidad hemos conocido qué son las **imágenes Drawable**, así como los distintos tipos de que disponemos. También hemos aprendido cómo trabajar con ellas en nuestras aplicaciones.
- Hemos analizado distintas **herramientas de Android Studio**, que nos facilitan el trabajo con esos elementos.
- Hemos visto qué son los **Styles** y **Themes**, para qué sirve cada uno y cuáles son sus principales **diferencias**. También hemos conocido el concepto de **herencia** y cómo un nuevo **style** puede heredar las propiedades de un **style** existente.
- Y, para finalizar, hemos aprendido a aplicar **styles** y **themes** a nuestro diseño de una aplicación, para **asignar estilos a nuestros proyectos**.

