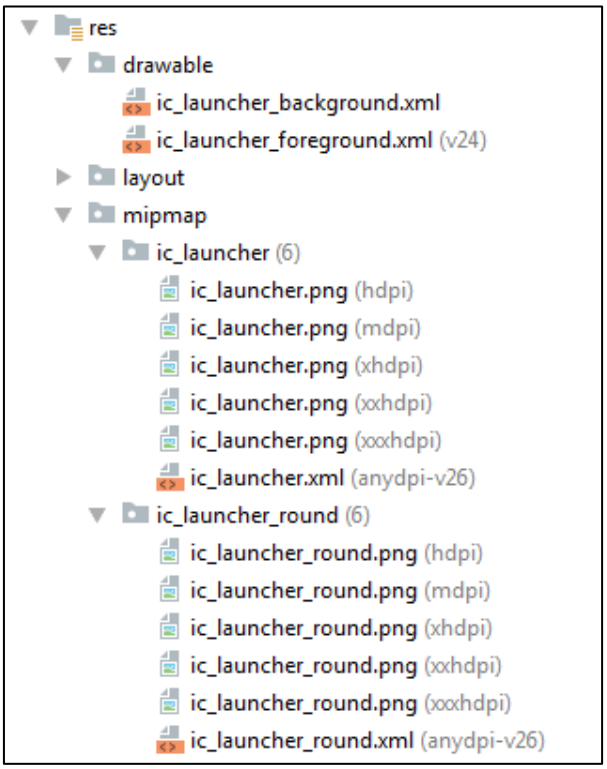
PRACTICA 2: EJEMPLOGRAFICOS 2

Parte VI: *Dibujar un BitmapDrawable de los recursos*

Busca en Internet un fichero gráfico en codificación *png* o *jpg* (los formatos gráficos usados por defecto en Android).

Paso 1. Renombra el fichero para que se llame *mi_imagen.png* o *mi_imagen.jpg* y arrastra a *res/drawable*. Podemos coger la imagen por defecto ic_launcher.png.



Paso 2. Declara la variable miImagen en la clase EjemploView del ejercicio anterior:

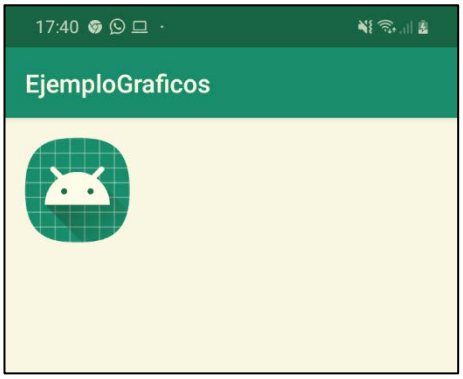
```
public class EjemploView extends View {

    private Drawable miImagen;
```

Paso 3. Escribe las siguientes tres líneas dentro del constructor de esta clase:

```
public EjemploView (Context context) {
    super(context);

    miImagen = ContextCompat.getDrawable(context, R.mipmap.ic_launcher);
    miImagen.setBounds( left: 30, top: 30, right: 200, bottom: 200);
}
```

Paso 4. Escribe la siguiente línea en el método onDraw:

```
@Override
protected void onDraw(Canvas canvas) {

    miImagen.draw(canvas);

}
```



Parte VII: Dibujar un gráfico vectorial

Android Studio permite usar gráficos vectoriales incluso en versiones anteriores a la 5.0. Para resolver el problema de compatibilidad con versiones anteriores, Android Studio detecta que estamos usando una versión mínima inferior a la 5.0; va a convertir los gráficos vectoriales a png. Veamos en este ejercicio como hacerlo.

Paso 1. Abre el proyecto del ejercicio anterior. Pulsa con el botón derecho sobre res/drawable y selecciona New/Drawable resource file. Introduce como nombre estrella.xml. Reemplaza su contenido por el que se acaba de mostrar

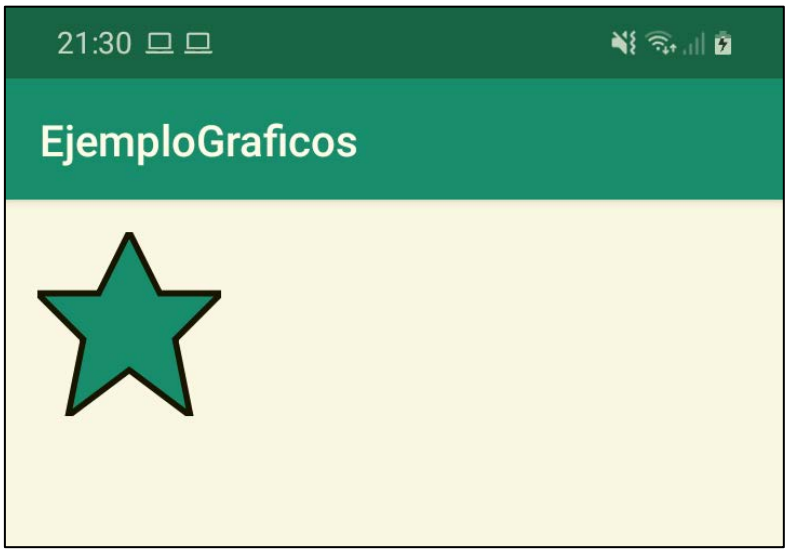
estrella.xml

```
<?xml version="1.0" encoding="utf-8"?>
<vector
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="60dp"
    android:height="60dp"
    android:viewportHeight="60"
    android:viewportWidth="60">
    <path
        android:fillColor="@color/colorPrimary"
        android:strokeColor="#000000"
        android:strokeWidth="2"
        android:pathData="M0 20 L20 20 L30 0 L40 20 L60 20 L45 35 L50 60 L30 45 L10 60 L15 35 Z" />
</vector>
```

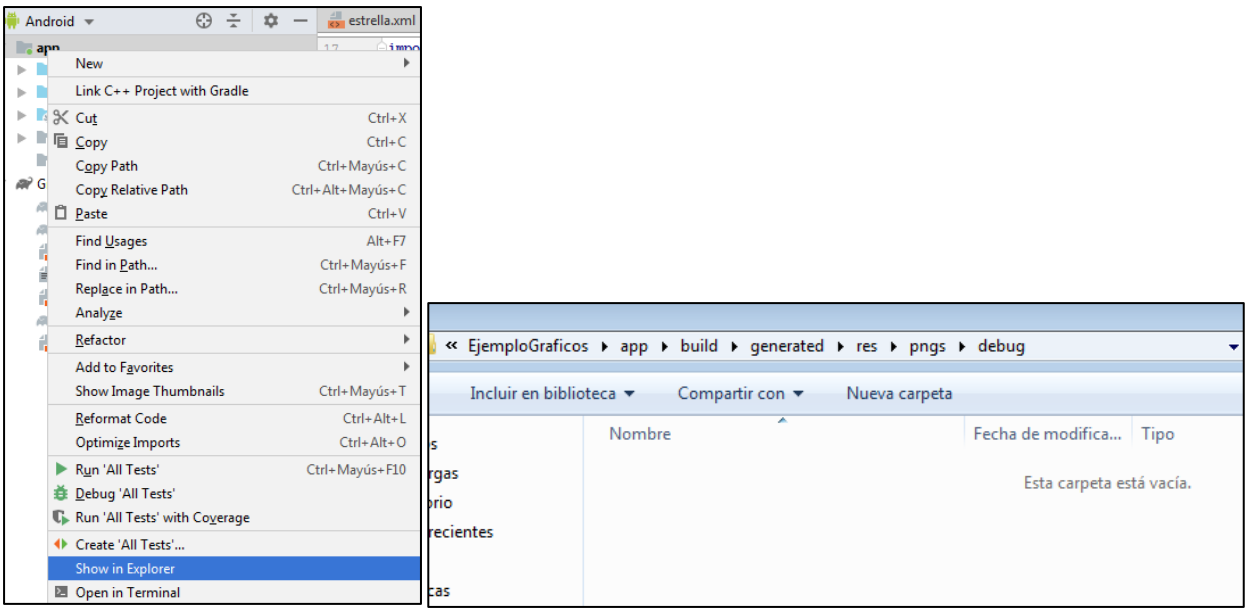
Paso 2. En la clase EjemploView del ejercicio anterior, haz el siguiente cambio:

```
public EjemploView (Context context) {
    super(context);

    //miImagen = ContextCompat.getDrawable(context, R.mipmap.ic_launcher);
    miImagen= ContextCompat.getDrawable(context, R.drawable.estrella);
    miImagen.setBounds( left: 30, top: 30, right: 200, bottom: 200);

}
```

Paso 3. Ejecuta la aplicación y verifica el resultado.



Paso 4. En el explorador del proyecto pulsa con el botón derecho sobre app y selecciona Show in explorer. Dentro de la carpeta del proyecto accede a app / build / generated / res / pngs / debug.



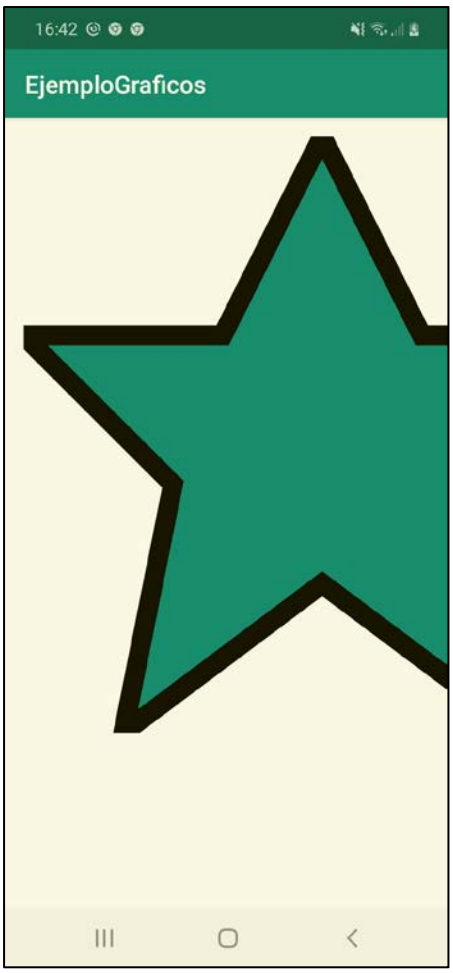
Observa como en la carpeta drawable-anydpi-v21 se guarda el fichero XML. Este recurso será usado en versiones API 21 o superior, para cualquier densidad gráfica En el resto de carpetas se ha generado un .png de forma automática. Este recurso será usado en versiones anteriores a la 21, según la densidad del dispositivo. Es conveniente definir el fondo de una vista en su *Layout* en XML en lugar de hacerlo por código. Comenta la línea introducida en el punto anterior e introduce el siguiente atributo en el *Layout main.xml*.

NOTA: *En caso de que el proyecto especifique una versión mínima de API igual o superior a la 21, estos pngs ya no serán generados.*

Paso 5. El gráfico ha sido diseñado para ser dibujado con 20 dp. Y esta información es la que utiliza el sistema para generar los pngs. Veamos qué pasa si lo dibujamos a un tamaño mucho mayor. Para ello cambia la siguiente línea:

```
public EjemploView (Context context) {
    super(context);

    //miImagen = ContextCompat.getDrawable(context, R.mipmap.ic_launcher);
    miImagen= ContextCompat.getDrawable(context, R.drawable.estrella);

    miImagen.setBounds( left: 30, top: 30, right: 1000, bottom: 1000);
}
```

Paso 6. Ejecuta la aplicación en un dispositivo con nivel de API 21 o superior. Observa como el reescalado es perfecto.



Parte VIII: Usar la librería de compatibilidad para VectorDrawable

Como acabamos de ver, Android Studio convierte automáticamente los gráficos vectoriales a png para que puedan ser utilizados en versiones anteriores a la 5.0. Esta solución tiene sus limitaciones, no podemos aumentar el tamaño, cambiar los atributos del gráfico dinámicamente (como uno de sus colores) y, además, ocupan más memoria. En este ejercicio, usaremos otra alternativa: usar una librería de compatibilidad.

Paso 1. Para usar gráficos vectoriales en versiones anteriores al API 21, añade al fichero build.gradle(Module:app):

```
android {
    defaultConfig {
        ...
        vectorDrawables.useSupportLibrary = true
    }
}
dependencies {
    ...
    implementation 'androidx.appcompat:appcompat:1.1.0'
}
```

NOTA: *La primera línea indica a Android Studio que vamos a utilizar la librería de compatibilidad para Vector Drawable. La segunda línea es posible que ya esté añadida en el proyecto.*

Paso 2. Borra la carpeta app / build / generated. Selecciona la opción Build / Make Project. Observa como ahora ya no se ha generado la carpeta app / build / generated / res / pngs. Ya no es necesario que se conviertan los ficheros vectoriales en png. Ahora, se podrá hacer en tiempo de ejecución por

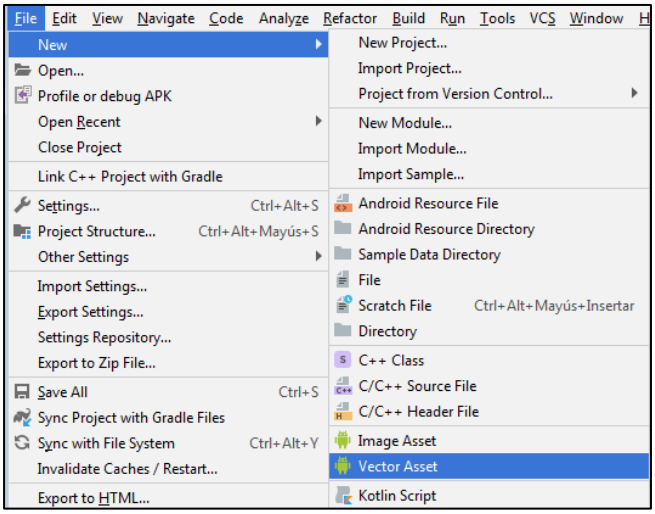
software.

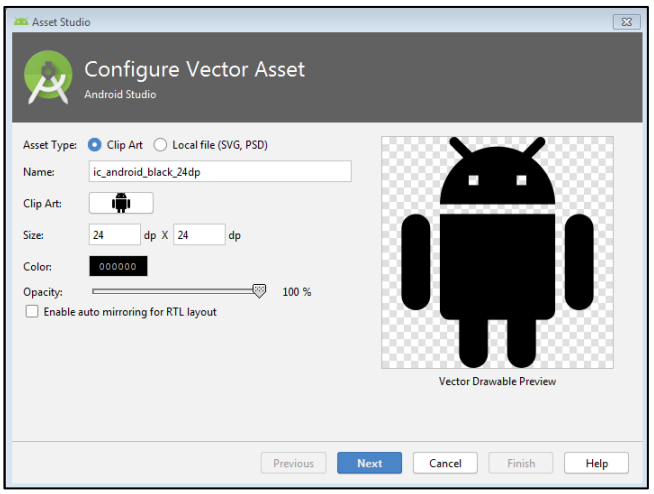
Paso 3. Ejecuta la aplicación en un dispositivo con nivel de API inferior a 21. Observa como el reescalado es perfecto.

Parte IX: Agregar gráficos vectoriales con Vector Asset

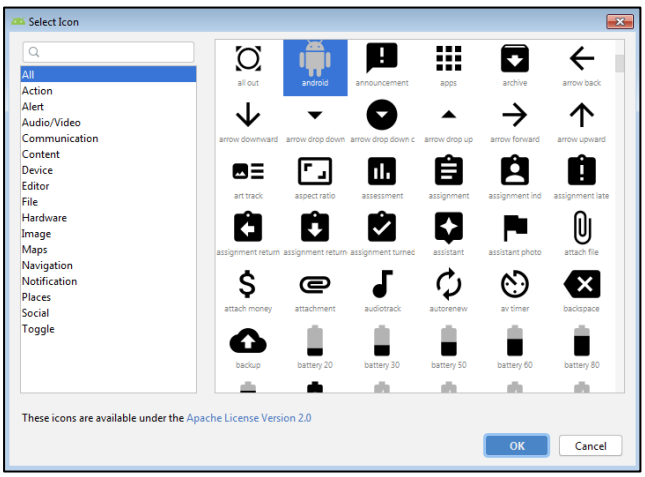
En el primer ejercicio de esta sección vimos cómo crear VectorDrawable mediante su código XML. En la práctica esto no se suele así así. Cuando necesitamos un gráfico podemos buscarlo en una colección, o si tenemos que generarlo nosotros usaremos un editor de gráficos vectoriales. Para ayudarnos en este trabajo, Android Studio incorpora la herramienta Vector Asset. Veamos cómo se usa en este ejercicio.

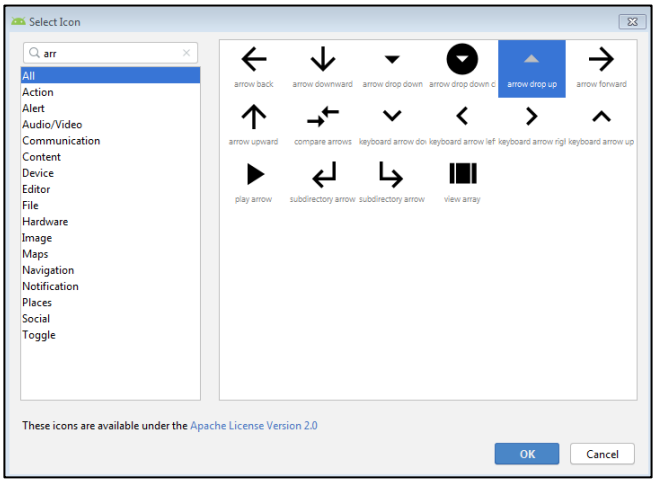
Paso 1. En Android studio selecciona *File / New / Vector Asset.*

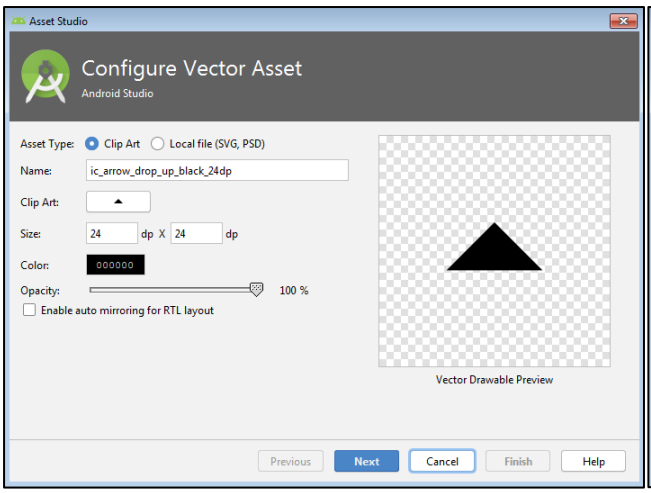


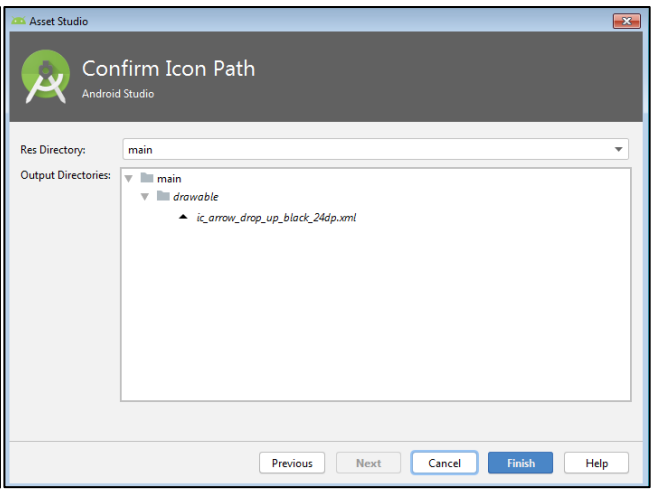


Paso 2. Seleccionando Clip Art, y pulsa en el icono. Puedes acceder a una colección de iconos. Selecciona uno muy sencillo, por ejemplo, arrow_drop_up. Pulsa Next y luego Finish.



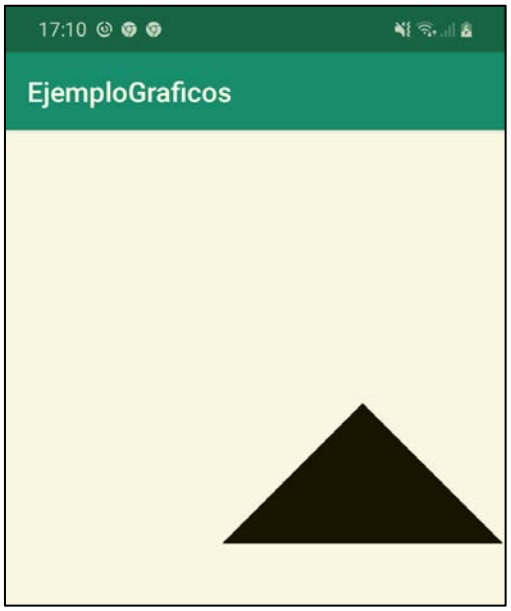






Paso 3. Modifica el ejercicio anterior para que se visualice este gráfico.

```
public EjemploView (Context context) {
    super(context);

    //miImagen = ContextCompat.getDrawable(context, R.mipmap.ic_launcher);
    miImagen= ContextCompat.getDrawable(context, R.drawable.ic_arrow_drop_up_black_24dp);

    miImagen.setBounds( left: 30, top: 30, right: 1000, bottom: 1000);

}
```



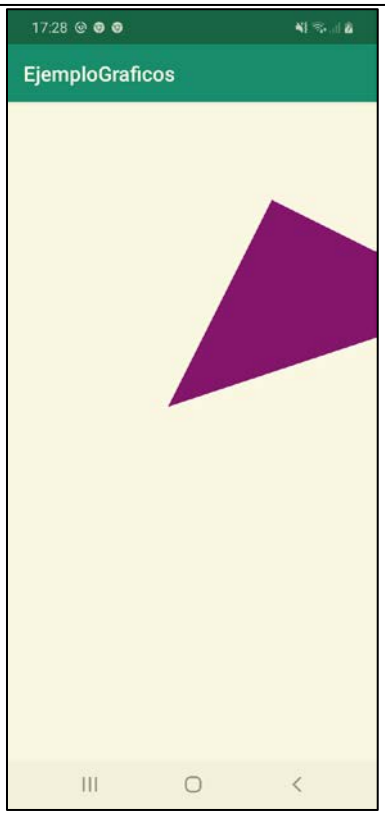
Paso 4. Abre el fichero XML que acabas de añadir. Observa con que poca información se ha definido el gráfico. Cambia el color y algún valor del path. Verifica que el resultado es el esperado.

ic_arrow_drop_up_black_24dp.xml

```xml
<vector xmlns:android="http://schemas.android.com/apk/res/android"
        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24.0"
        android:viewportHeight="24.0">
    <path
        android:fillColor="#FF000000"
        android:pathData="M7,1415,-5 5,5z"/>
</vector>
```

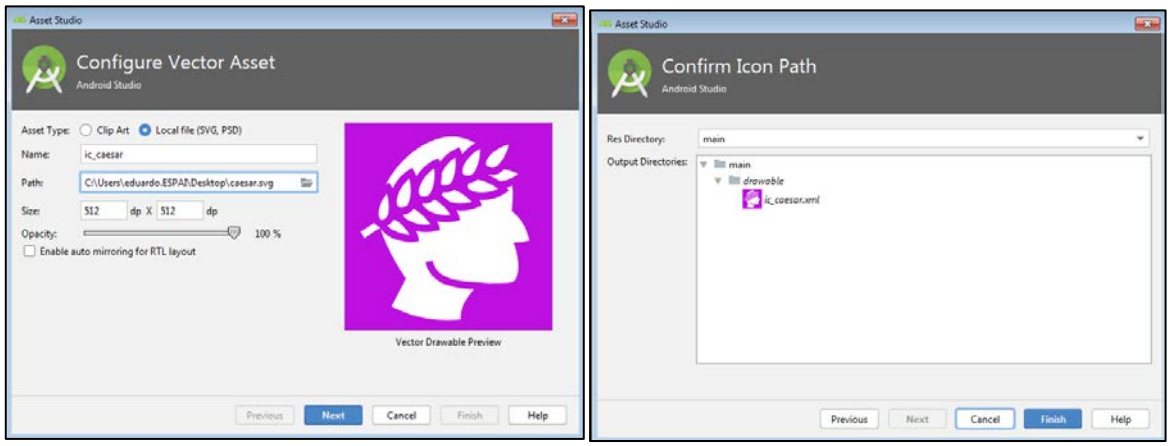
ic_arrow_drop_up_black_24dp.xml

```xml
<vector xmlns:android="http://schemas.android.com/apk/res/android"
        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24.0"
        android:viewportHeight="24.0">
    <path
        android:fillColor="#FF770077"
        android:pathData="M7,1415,-10 10,5z"/>
</vector>
```



Paso 5. Busca en Internet y fichero con extensión SVG (Scalable Vector Graphics) o PSD (Adobe Photoshop). NOTA: Se pueden encontrar aquí: https://game-icons.net

Paso 6. Abre de nuevo Vector Asset y selecciona Local file (SVG, PSD). En el campo Name indica un nombre de recurso y en Path el fichero que acabas de descargar. Si lo deseas también puedes cambiar el tamaño que tendrá por defecto la imagen y el grado de opacidad.



Paso 7. El formato XML usado en VectorDrawable no soporta todas las características disponibles en los formatos SVG y PSD. En el cuadro Errors, que encontrarás en la parte inferior, se mostrará un listado con los problemas en la conversión.

Paso 8. Observa como convierte el fichero al formato utilizado en Android.

ic_caesar.xml

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="512dp"
    android:height="512dp"
    android:viewportWidth="512"
    android:viewportHeight="512">
  <path
      android:pathData="M0,0h512v512H0z"
      android:fillColor="#bd10e0"
      android:fillAlpha="1"/>
  <path
      android:pathData="M264.234,33.64a94.945,94.945 0,0 0,-10.957 0.608C190.895,41.376 131.82
      android:fillColor="#fff"
      android:fillAlpha="1"/>
</vector>
```

La gran ventaja de usar gráficos vectoriales es que ocupan muy poco espacio, podemos cambiar fácilmente los colores y no tenemos que preocuparnos de preparar diferentes recursos para diferentes resoluciones. Como inconveniente los gráficos vectoriales pueden costar más recursos para ser generados, especialmente si son complejos y tienen curvas.

Parte X: *Definir un ShapeDrawable*

Veamos un ejemplo de cómo utilizar un objeto ShapeDrawable para crear una vista a medida.

Paso 1. Abre el proyecto *EjemploGraficos*.

Paso 2. En la clase EjemploView declara la siguiente variable:

```
public class EjemploView extends View {

    //private Drawable miImagen;
    private ShapeDrawable miImagen;
```

Paso 3. Añade las siguientes tres líneas dentro del constructor de esta clase:

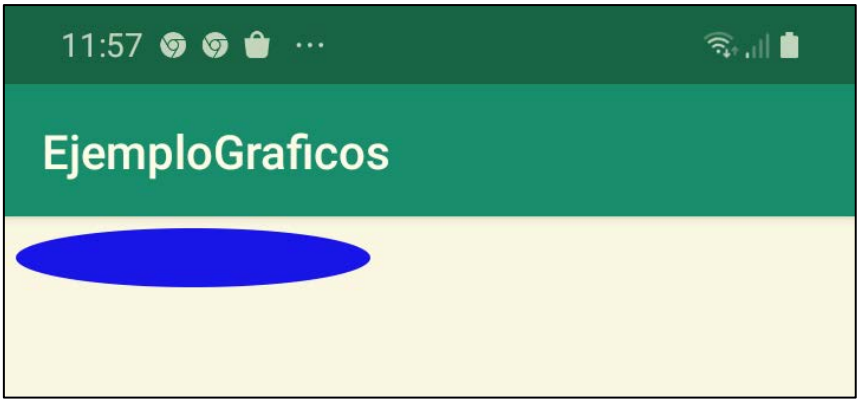
```
public EjemploView (Context context) {
    super(context);

    miImagen  = new ShapeDrawable(new OvalShape());
    miImagen.getPaint().setColor(0xff0000ff);
    miImagen.setBounds( left: 10,  top: 10,  right: 310,  bottom: 60);

    //miImagen = ContextCompat.getDrawable(context, R.mipmap.ic_launcher);
    //miImagen= ContextCompat.getDrawable(context, R.drawable.ic_caesar);
    //miImagen.setBounds(30,30,1000,1000);
}
```

En el constructor, un objeto ShapeDrawable es definido como un óvalo. Se le asigna un color y se definen sus fronteras.

Paso 4. Escribe la siguiente línea en el método onDraw:

```
@Override
protected void onDraw(Canvas canvas) {

    miImagen.draw(canvas);

}
```

Paso 5. Ejecuta la aplicación y observa el resultado.



Parte XI: Creación de una nueva vista independiente

En este ejercicio vamos a poner la clase EjemploView en un fichero independiente para que pueda ser utilizada desde cualquier parte.

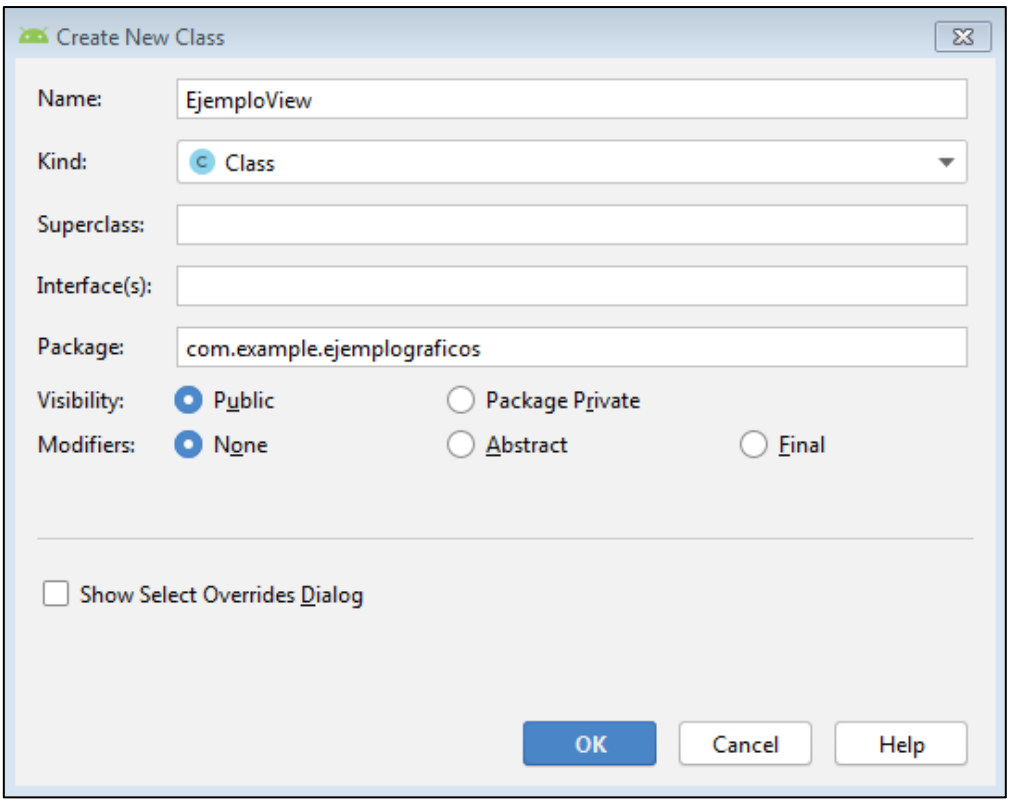
Paso 1. Abre el proyecto *EjemploGraficos*.

Paso 2. En la clase EjemploGraficosActivity selecciona todo el texto correspondiente a la definición de la clase EjemploView y córtalo al portapapeles.

```
EjemploGraficosActivity.java
        setContentView(new EjemploView( context: this));
    }

    public class EjemploView extends View {

        //private Drawable miImagen;
        private ShapeDrawable miImagen;

        public EjemploView (Context context) {
            super(context);

            miImagen  = new ShapeDrawable(new OvalShape());
            miImagen.getPaint().setColor(0xff0000ff);
            miImagen.setBounds( left: 10,  top: 10,  right: 310,  bottom: 60);

            //miImagen = ContextCompat.getDrawable(context, R.mipmap.ic_launcher);
            //miImagen= ContextCompat.getDrawable(context, R.drawable.ic_caesar);
            //miImagen.setBounds(30,30,1000,1000);
        }

        //miImagen= VectorDrawableCompat.create(getContext().getResources(), R.drawable.estrella,null);
        //miImagen.setBounds(0, 0, miImagen.getIntrinsicWidth(), miImagen.getIntrinsicHeight());

        @Override
        protected void onDraw(Canvas canvas) {

            miImagen.draw(canvas);

        }
```

Paso 3. Pulsa con el botón derecho sobre *java/com.example.ejemplograficos* y selecciona la opción *New/Class...*

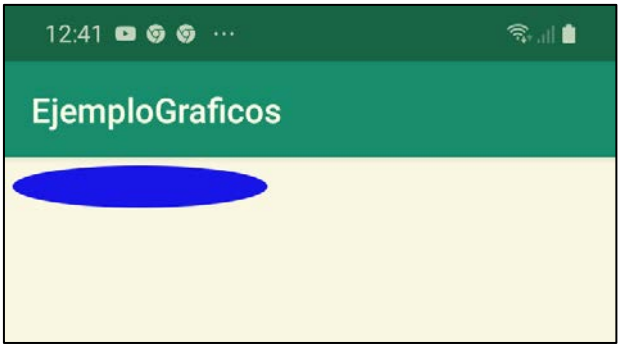
Paso 4. Introduce como nombre de la clase EjemploView.



Paso 5. Pega el texto que has puesto en el portapapeles.

```
public class EjemploView extends View {

    private ShapeDrawable miImagen;

    public EjemploView (Context context) {
        super(context);
        miImagen  = new ShapeDrawable(new OvalShape());
        miImagen.getPaint().setColor(0xff0000ff);
        miImagen.setBounds( left: 10,  top: 10,  right: 310,  bottom: 60);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        miImagen.draw(canvas);
    }
}
```

Paso 6. Verifica que al ejecutar la aplicación el resultado es idéntico al obtenido en la sección anterior.



En este apartado aprovechamos para introducir otros aspectos que tendrás que tener en cuenta para crear nuevas vistas. Cuando quieres crear una nueva vista, tendrás que extender la clase View, escribir un constructor y como mínimo sobreescribir los métodos onSizeChanged() y onDraw(). Es decir,

tendrás que seguir el siguiente esquema:

```
public class EjemploView extends View {

    public EjemploView (Context context, AttributeSet attrs) {
        super(context, attrs);
        //Inicializa la vista
        //OjO: Aún no se conocen sus dimensiones
    }

    @Override protected void onSizeChanged(int ancho, int alto,
                                           int ancho_anterior, int alto_anterior){
        //Te informan del ancho y el alto
    }

    @Override protected void onDraw(Canvas canvas) {
        //Dibuja aquí la vista
    }

}
```

Observa como el constructor utilizado tiene dos parámetros: El primero de tipo Context te permitirá acceder al contexto de aplicación, por ejemplo para utilizar recursos de esta aplicación. El segundo, de tipo AttributeSet, te permitirá acceder a los atributos de esta vista, cuando sea creada desde XML. El constructor es el lugar adecuado para crear todos los componentes de tu vista, pero cuidado, en este punto todavía no se conoce las dimensiones que tendrá.

Android realiza un proceso de varias pasadas para determinar el ancho y alto de cada vista dentro de un *Layout*. Cuando finalmente ha establecido las dimensiones de una vista llamará a su método onSizeChanged(). Nos indica como parámetros el ancho y alto asignado. En caso de tratarse de un eajuste de tamaños, por ejemplo una de las vistas del Layout desaparece y el resto tienen que ocupar su espacio, se nos pasará el ancho y alto anterior. Si es la primera vez que se llama al método estos parámetros valdrán 0.

El último método que siempre tendrás que reescribir es onDraw(). Es aquí donde tendrás que dibujar la vista.

Parte XII: Creación de una vista que pueda ser diseñada desde XML

Vamos a modificar la vista anterior para que pueda ser utilizada usando un diseño en XML:

Paso 1. Modifica el código de la clase EjemploView para que coincida con el siguiente (en negrita se resaltan las diferencias):

```
public class EjemploView extends View {
    private ShapeDrawable miImagen;
    public EjemploView (Context context, AttributeSet attrs) {
        super(context, attrs);
        miImagen = new ShapeDrawable(new OvalShape());
        miImagen.getPaint().setColor(0xff0000ff);
    }

    @Override protected void onSizeChanged(int ancho, int alto,
                                           int ancho_anterior, int alto_anterior){
        miImagen.setBounds( left: 0,  top: 0, ancho, alto);
    }

    @Override protected void onDraw(Canvas canvas) {
        miImagen.draw(canvas);
    }
}
```

La primera diferencia es la utilización de un constructor con el parámetro AttributeSet. Este constructor es imprescindible si quieres poder definir la vista en XML. También se ha añadido el método onSizeChanged, para que el óvalo se dibuje siempre ajustado al tamaño de la vista.

Paso 2. Abre el fichero activity_main.xml y reemplaza su contenido por el siguiente:

activity_main.xml

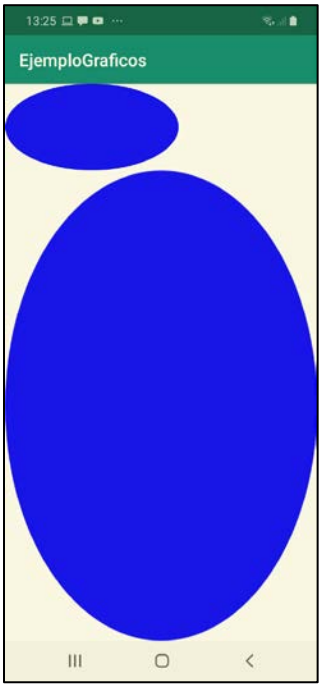
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <com.example.ejemplograficos.EjemploView
        android:id="@+id/ejemploView1"
        android:layout_width="400px"
        android:layout_height="200px" />
    <com.example.ejemplograficos.EjemploView
        android:id="@+id/ejemploView2"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

NOTA: *No utilices el valor "wrap_content". Nuestra vista no ha sido programada para soportar este tipo de valor*

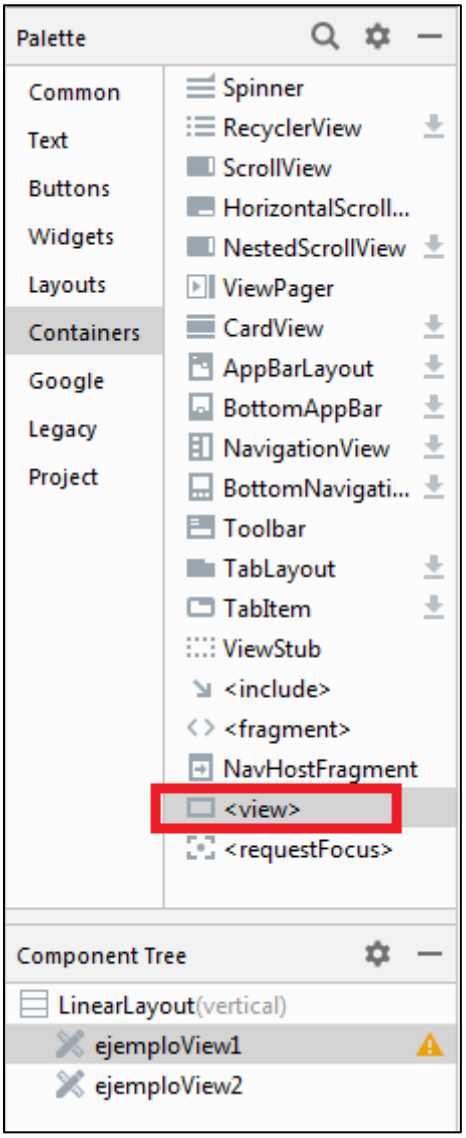
Paso 3. En la clase EjemploGraficosActivity reemplaza setContentView(new EjemploView(this)) por setContentView(R.layout.main).

```
public class EjemploGraficosActivity extends AppCompatActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(new EjemploView(this));
        setContentView(R.layout.activity_main);
    }
```

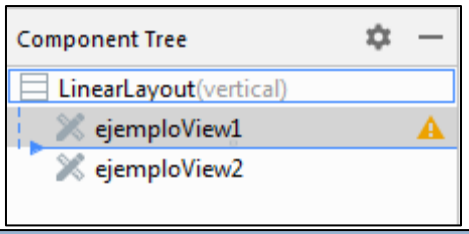
Paso 4. Ejecuta la aplicación. El resultado ha de ser similar a:

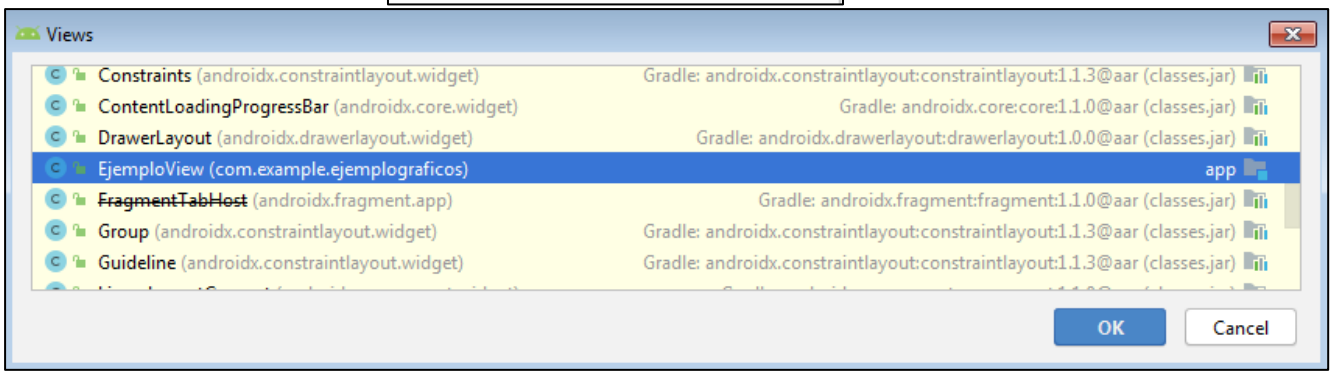


Paso 5. Esta nueva vista también puede ser insertada en un layout usando el editor visual. Pulsa en la lengüeta Design para pasar a este modo de edición. En la paleta de vistas, busca la sección Containers:

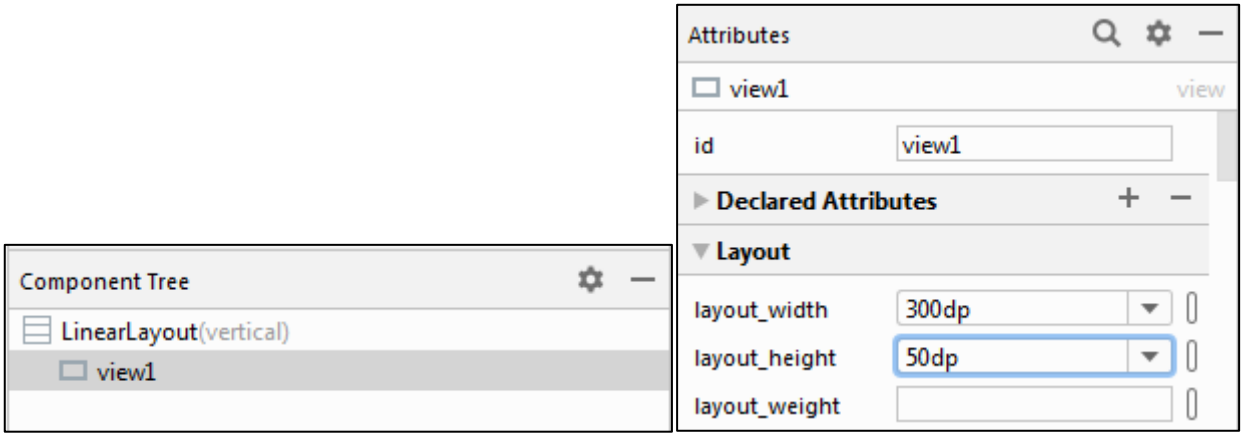


Paso 6. Selecciona <view> y arrástralo entre los dos óvalos. A continuación te pedirá que indiques la clase a utilizar. Selecciona EjemploView.

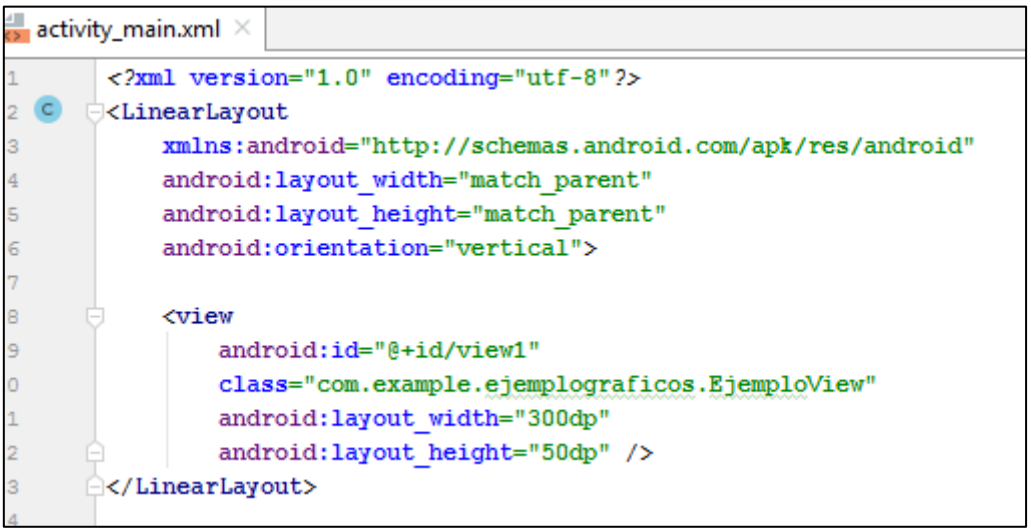




Paso 7. Modifica en la ventana de Properties los siguientes atributos:



Paso 8. Selecciona la lengüeta Text para observar el código introducido:



Paso 9. Compáralo con el código introducido en el punto 2. Se trata dos formas alternativas para introducir una vista creada por ti.

```
<view
    android:id="@+id/view1"
    class="com.example.ejemplograficos.EjemploView"
    android:layout_width="300dp"
    android:layout_height="50dp" />
```

```
<com.example.ejemplograficos.EjemploView
    android:id="@+id/ejemploView2"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Paso 10. Ejecuta la aplicación. El resultado ha de ser similar a: