Práctica 18: Drawable

Objetivo:

- Conocer y trabajar con la clase Drawable

Tiempo destinado: 2hrs.

Fundamentos:

La clase Drawable es una abstracción que representa «algo que se puede dibujar». Esta clase se extiende para definir una gran variedad de objetos gráficos más específicos. Muchos de ellos pueden definirse como recursos usando ficheros XML. Entre ellos tenemos los siguientes:

- **BitmapDrawable**: Imagen basada en un archivo gráfico (PNG o JPG). Etiqueta XML <bitmap>.
- **VectorDrawable**: Permite crear gráficos de forma vectorial. Podemos definir los gráficos en XML usando un formato basado en SVG (Scalable Vector Graphics). Este Drawable está disponible desde Android v5.0 (API 21), no obstante, se ha incluido en la librería de compatibilidad v7. Por lo tanto, podremos utilizarlo desde el API 7.
- **LayerDrawable**: Contiene un array de Drawable que se visualizan según el orden del array. El índice mayor del array es el que se representa encima. Cada Drawable puede situarse en una posición determinada. Etiqueta XML <layerlist>.
- **GradientDrawable**: Degradado de color que se puede ser usado en botones o fondos.

  Etiqueta XML <gradient>.
- **AnimationDrawable**: Permite crear animaciones frame a frame a partir de una serie de objetos Drawable. Etiqueta XML <animation-list>.

También puede ser interesante que uses la clase Drawable o uno de sus descendientes como base para crear tus propias clases gráficas. Además de ser dibujada, la clase Drawable proporciona una serie de mecanismos genéricos que permiten indicar cómo ha de ser dibujada. No todo Drawable ha de implementar todos los mecanismos. Veamos los más importantes:

- **setBounds(x1,y1,x2,y2)** permite indicar el rectángulo donde ha de ser dibujado. Todo Drawable debe respetar el tamaño solicitado por el cliente, es decir, ha de permitir el escalado. Podemos consultar el tamaño preferido de un Drawable mediante los métodos getIntrinsicHeight() y getIntrinsicWidth().

- **getPadding(Rect)** proporciona información sobre los márgenes recomendados para representar el contenido. Por ejemplo, un Drawable que intente ser un marco para un botón debe devolver los márgenes correctos para localizar las etiquetas, u otros contenidos, en el interior del botón.

- **setState(int[])** permite indicar al Drawable en qué estado ha de ser dibujado; por ejemplo, «con foco», «seleccionado», etc. Algunos Drawable cambiarán su representación según este estado.

- **setLevel(int)** algunos Drawable tienen un nivel asociado. Con este método podemos cambiarlo. Por ejemplo, un nivel puede interpretarse como una batería de niveles o un nivel de progreso.

En esta práctica exploraremos el uso de elementos Drawable.

Instrucciones:

1. Realice correctamente cada paso en la sección de desarrollo, documente la práctica con capturas de pantalla, mostrando cada paso realizado.

2. Suba su reporte al espacio asignado en el aula virtual.

Desarrollo:

1. Genere un nuevo proyecto con las mismas características de los proyectos anteriores, con el nombre de: "**practica18**":
   a. Phone and Tablet, Empty Activity
   b. Nombre: **practica18**, Language: **kotlin**
   c. Min SDK: **Api 21**

2. BitmapDrawable.
   a. Agregue una imagen a la sección de recursos (res/drawable) con el nombre de Android.png

   b. Genere una nueva clase kotlin con el siguiente contenido:

```
class MyCanvasView(context: Context) : View(context) {

    private lateinit var miImagen: Drawable

    override fun onSizeChanged(width: Int, height: Int, oldWidth: Int, oldHeight: Int) {
        super.onSizeChanged(width, height, oldWidth, oldHeight)
        miImagen= AppCompatResources.getDrawable(context,R.drawable.android)!!

        miImagen.setBounds( left: 30, top: 30, right: 200, bottom: 200)
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        miImagen.draw(canvas)

    }
}
```

c. Modifique la actividad principal, para tener el siguiente código en la función OnCreate:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    //setContentView(R.layout.activity_main)
    setContentView( MyCanvasView( context: this))
}
```

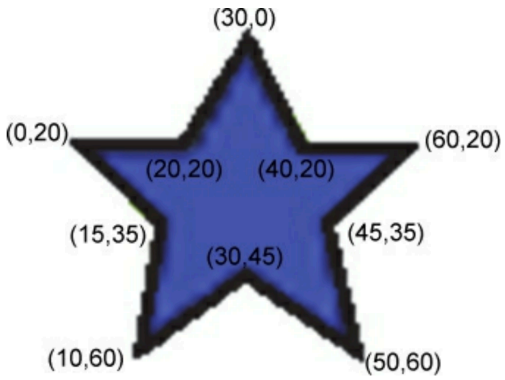
d. Ejecute y pruebe la aplicación.

e. Modifique las dimensiones para observar como cambia de tamaño la imagen. (miImagen.setBounds de la clase MyCanvasView)

3. VectorDrawable

a. Genere un nuevo recurso drawable como un xml, para ello genere el archivo estrella.xml en la carpeta /res/drawable

```
<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>
```

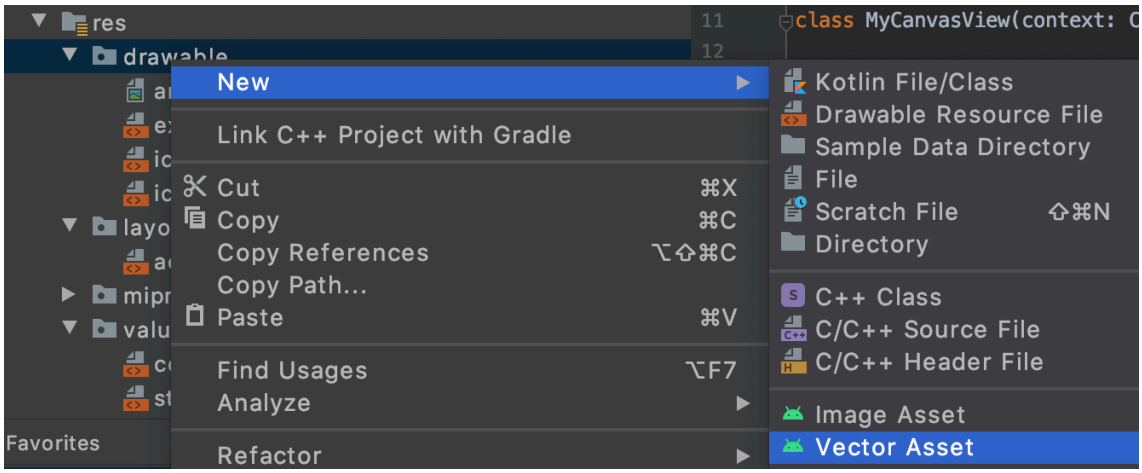


b. Modifique la clase MyCanvasView para llamar al nuevo recurso que creamos:

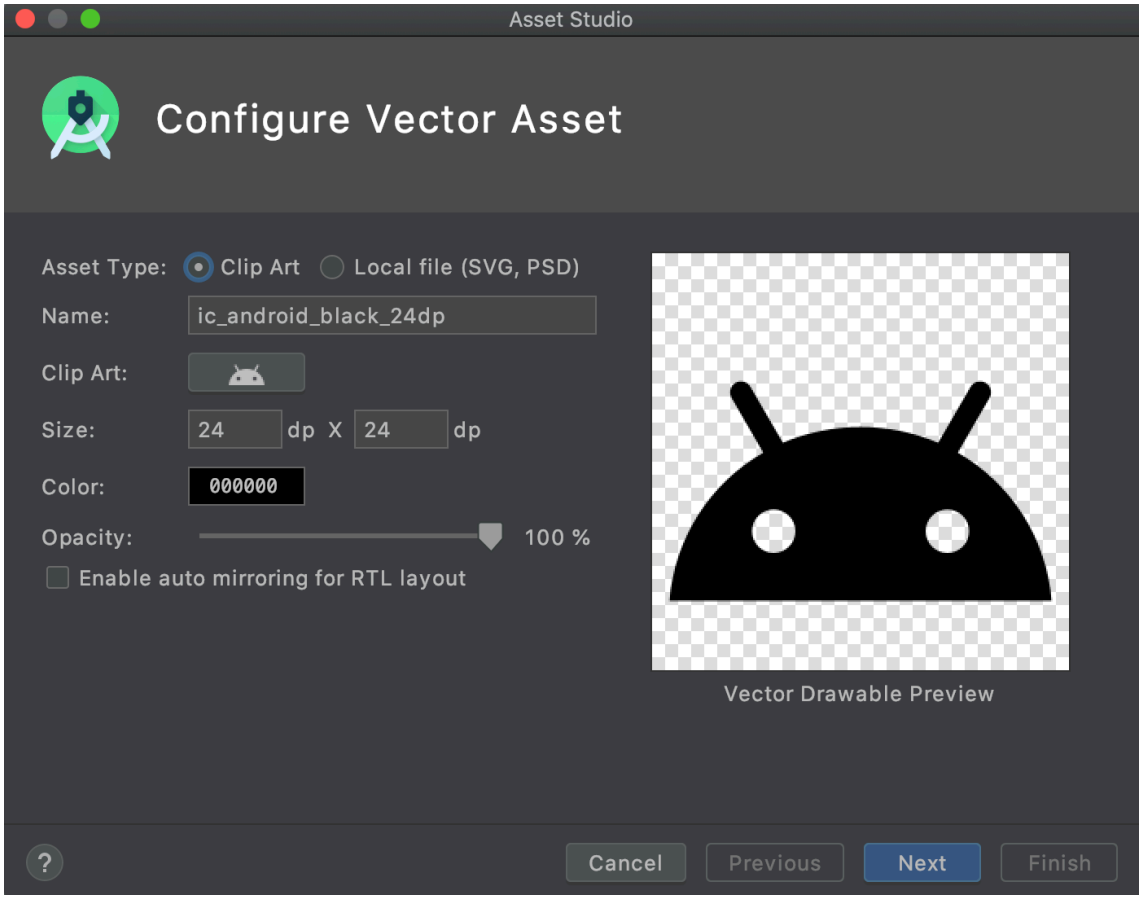
```
override fun onSizeChanged(width: Int, height: Int, oldWidth: Int, oldHeight: Int) {
    super.onSizeChanged(width, height, oldWidth, oldHeight)
    // miImagen= AppCompatResources.getDrawable(context,R.drawable.android)!!
    miImagen= AppCompatResources.getDrawable(context,R.drawable.extrella)!!
    miImagen.setBounds( left: 30, top: 30, right: 200, bottom: 200)
}
```

c. Este grafico fue creado para mostrarse con un tamaño de 200x200, modifique las dimensiones y observe como se visualiza la estrella.

4. Vector Drawable de un archivo
   a. En la practica es muy complicado dejar la parte de diseño al programador, por lo que Android studio da la posibilidad de importar imágenes vectoriales y poder utilizarlas, para ello seleccione file->new-> Vector Asset



b. Aparecerá la siguiente pantalla en la que nos da la opción de importar por medio de una galería de iconos de Android Studio o seleccionar una imagen vectorizada:



c. Explore los iconos del Clip Art de Android y seleccione alguno, finalice y vea el archivo resultante.

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24"
        android:viewportHeight="24">
    <path
        android:pathData="M17.6,11.48 L19.44,8.3a0.63,0.63 0,0 0,-1.09 -0.63l-1.88,3.24a11.43,11.43
        android:fillColor="#FF000000"/>
</vector>
```

d. Pruebe el nuevo recurso y visualice en su dispositivo.

5. GradientDrawable
   a. Genere un nuevo gradiente.xml en /res/drawable con lo siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient
        android:startColor="#FFFFFF"
        android:endColor="#0000FF"
        android:angle="270" />
</shape>
```

Observe que se indica el color inicial y final, ademas del angulo del degradado (solo se permite 0, 90, 180 y 270).

b. Agregue el nuevo degradado como fondo en la clase MyCanvasView

```
override fun onSizeChanged(width: Int, height: Int, oldWidth: Int, oldHeight: Int) {
    super.onSizeChanged(width, height, oldWidth, oldHeight)
    // miImagen= AppCompatResources.getDrawable(context,R.drawable.android)!!
    setBackgroundResource(R.drawable.degradado)
    miImagen= AppCompatResources.getDrawable(context,R.drawable.extrella)!!
    miImagen.setBounds( left: 30, top: 30, right: 200, bottom: 200)
}
```

Tambien es posible definir el degradado creado como fondo en un xml:

```
<?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"
    android:background="@drawable/degradado">

    <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>
```

6. AnimationDrawable
    a. Genere el siguiente archivo animación.xml, en la carpeta /res/drawable, con el siguiente contenido:

```xml
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android" android:oneshot="false">
    <item android:drawable="@drawable/android_2"
          android:duration="100" />
    <item android:drawable="@drawable/android_3"
        android:duration="100" />
    <item android:drawable="@drawable/android_4"
        android:duration="100" />
</animation-list>
```

    b. Agregue las imágenes listadas en el archivo anterior (android_2.png, android_3.png y android_4.png) en los recursos (/res/drawable)

    c. Modifique el código de la clase principal, para permitir visualizar la animación:

```kotlin
class MainActivity : AppCompatActivity() {


    private lateinit var animacion: AnimationDrawable

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        //setContentView(R.layout.activity_main)
        //setContentView( MyCanvasView(this))
        animacion = AppCompatResources.getDrawable( context: this, R.drawable.animacion) as AnimationDrawable
        val vista = ImageView( context: this)
        vista.setBackgroundColor(Color.WHITE)
        vista.setImageDrawable(animacion)
        animacion.start()
        setContentView(vista)

    }
}
```

7. Redacte sus observaciones y conclusiones, poniendo principal énfasis en los elementos nuevos.