

Programación multimedia y dispositivos móviles

UD04. Recursos de una aplicación

Desarrollo de Aplicaciones Multiplataforma



Anna Sanchis Perales

ÍNDICE

1. OBJETIVOS	3
2. RECURSOS	4
3. AGRUPAMIENTO DE RECURSOS	5
4. ACCESO A LOS RECURSOS	8
5. RECURSOS ALTERNATIVOS	9
5.1. Diferentes layouts (horizontal y vertical)	9
5.2. Traducción de una aplicación	18
5.3 Normas en la forma de nombrar recursos con sus sufijos	21
5.4. Icono para lanzar la aplicación	27
5.4.1. Generar icono con todas las densidades	27
6. RECURSOS DEL SISTEMA	30
7. ESTILOS Y TEMAS	32
7.1. Estilos	32
7.2. Heredar de un estilo propio	33
7.3. Los temas	37
8. ESCUCHADORES DE EVENTOS EN ANDROID	39
9. ANIMACIONES LOTTIE EN APLICACIONES ANDROID	41

1. OBJETIVOS

Los objetivos a conocer y comprender son:

- Mostrar cómo se utilizan los recursos alternativos.
- Aprenderemos a crear estilos y temas para personalizar nuestras aplicaciones.
- Mostrar cómo interactuar con las vistas desde el código Kotlin.

2. RECURSOS

Una aplicació Android va a poder ser executada en una gran varietat de dispositius. El tamaño de pantalla, la resolució o el tip de entrades pot variar molt d'un dispositiu a otro. Per otra part, nostra aplicació ha de estar preparada per diferents modes de funcionament, com el mode "automóvil" o el mode "noche", y per poder executar-se en diferents idiomes.

A la hora de crear la interfaz de usuario, hemos de tener en cuenta todas estas circunstancias. Afortunadamente, la plataforma Android nos proporciona una herramienta de gran potencia para resolver este problema: **el uso de los recursos alternativos**.

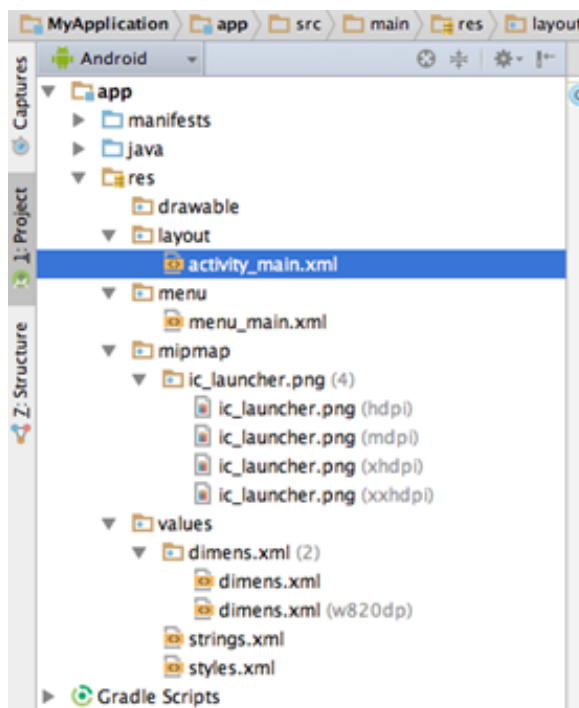
La definición de los recursos en Android es un aspecto muy importante en el diseño de una aplicación. Una de sus principales ventajas es que facilita a los diseñadores gráficos e introductores de contenido trabajar en paralelo con los programadores.

Es recomendable almacenar de forma externa los recursos como imágenes o cadenas de caracteres que vayas a utilizar en la aplicación respecto al código fuente de la misma. De esa manera puedes proporcionar recursos alternativos que soportan configuraciones específicas de los dispositivos, tales como diferentes lenguajes o tamaños de pantalla. Para proporcionar compatibilidad entre las diferentes configuraciones que ofrecen los dispositivos Android, debes organizar los recursos en la **carpeta /res** del proyecto, usando varias subcarpetas que agrupen los recursos por tipo y configuración.

Por lo que añadir un recurso a nuestra aplicación es muy sencillo, no tenemos más que añadir un fichero dentro de una carpeta determinada de nuestro proyecto. Para cada uno de los recursos que añadamos, el sistema crea, de forma automática, un id de recurso dentro de la **clase R**.

3. AGRUPAMIENTO DE RECURSOS

Cada tipo de recurso debe almacenarse en una subcarpeta específica dentro de la carpeta res del proyecto. Por ejemplo, en un proyecto sencillo se puede tener una estructura de recursos como la siguiente:



Según la carpeta que utilicemos, el recurso creado será de un tipo específico. Pasamos a enumerar las carpetas y tipos posibles:

Carpeta Identificador	Descripción
/res/drawable/ R.drawable	Ficheros en bitmap (.png, .jpg o .gif). Ficheros PNG en formato Nine-patch (.9.png). Ficheros XML con descriptores gráficos (ver clase Drawable).
/res/layout/ R.layout	Ficheros XML con los layouts usados en la aplicación.
/res/menu/ R.menu	Ficheros XML con la definición de menús. Podemos asignar una actividad o una vista.
/res/anim/ R.anim	Fichero XML que permiten definir animaciones Tween, también conocidas como animaciones de vista.

/res/animador/ R.animador	Ficheros XML que permiten modificar las propiedades de un objeto a lo largo del tiempo. Cuando veamos las animaciones de propiedades lo veremos.
/res/xml/ R.xml	Otros ficheros XML, como los ficheros de preferencias.
/res/raw/ R.raw	Ficheros que se encuentran en formato binario. Por ejemplo ficheros de audio o vídeo.
/res/values/ R.values	Ficheros XML que definen un determinado valor para definir un color, estilo, cadena de caracteres... se describen en la siguiente tabla.

Veamos los tipos de recursos que encontramos dentro de la carpeta **/res/values**:

Fichero por defecto Identificador	Descripción
strings.xml R.string	Identifica cadenas de caracteres <pre><string name="saludo">¡Hola Mundo!</string> <string name="btn_entrar">Entrar</string></pre>
colors.xml R.color	Un color definido en formato ARGB (alfa, rojo, verde y azul). Los valores se indican en hexadecimal en uno de los siguientes formatos: #RGB, #ARGB, #RRGGBB ó #AARRGGBB <pre><color name="verde_opaco">#0f0</color> <color name="red_translucido">#80ff0000</color></pre>
dimensions.xml R.dimen	Un número seguido de una unidad de medida. px - píxeles, mm - milímetros, in – pulgadas, pt – puntos (=1/72 pulgadas), dp – píxeles independientes de la densidad (=1/160 pulgadas), sp – igual que dp pero cambia según las preferencias de tamaño de fuente. <pre><dimen name="alto">2.2mm</dimen> <dimen name="tamano_fuente">16sp</dimen></pre>
styles.xml R.style	Definen una serie de atributos que pueden ser aplicados a una vista o a una actividad. Si se aplican a una actividad se conocen como temas. <pre><style name="TextoGrande"parent="@style/Text"> <item name="android:textSize">20pt</item> <item name="android:textColor">#000080</item> </style></pre>

R.int	<p>Define un valor entero.</p> <pre><integer name="max_asteroïdes">5</integer></pre>
R.bool	<p>Define un valor booleano.</p> <pre><bool name="misiles_ilimitados">true</bool></pre>
R.id	<p>Define un recurso de ID único. La forma habitual de asignar ID a los recursos es utilizando el atributo id="@+id/nombre". Aunque en algunos casos puede ser interesante disponer de IDs previamente creados, para que los elementos así nombrados tengan una determinada función. Este tipo de IDs se utilizan en las vistas TabHost y ListView.</p> <pre><item type="id" name="button_ok"/> <item type="id" name="dialog_exit"/></pre>
R.array	<p>Una serie ordenada de elementos. Pueden ser de strings, de enteros o de recursos (TypedArray)</p> <pre><string-array name="dias_semana"> <item>lunes</item> <item>martes</item> </string-array> <integer-array name="primos"> <item>2</item> <item>3</item> <item>5</item> </integer-array> <array name="asteroïdes"> <item>@drawable/asteroïde1</item> <item>@drawable/asteroïde2</item> </array></pre>

Aunque el sistema crea ficheros que aparecen en la columna de la izquierda de la tabla anterior y se recomienda definir los recursos de cadena dentro de strings.xml, hay que resaltar que no es más que una sugerencia de organización. Sería posible mezclar cualquier tipo de recurso de esta tabla dentro de un mismo fichero y poner a este fichero cualquier nombre.

4. ACCESO A LOS RECURSOS

Una vez definido un recurso, este puede ser utilizado desde un fichero XML o desde Kotlin.

A continuación se muestra un ejemplo desde **XML**:

```
<ImageView
    android:layout_height="@dimen/alto"
    android:layout_width="match_parent"
    android:background="@drawable/imagen"
    android:text="@string/saludo"
    android:text_color="@color/verde_opaco"/>
```

Para acceder a un recurso definido desde **Kotlin** usaremos el siguiente código (ir leyendo poco a poco el código y comprender lo que hace cada uno):

```
//EJEMPLO ACCESO A UN RECURSO IMAGEN
// Importa el paquete necesario si aún no lo has hecho
import android.content.Context
import android.graphics.drawable.Drawable

// Obtén una referencia al recurso Drawable
val drawableResource: Drawable? = context.getDrawable(R.drawable.my_image)

//EJEMPLO ACCESO A UNA CADENA STRING
// Importa el paquete necesario si aún no lo has hecho
import android.content.Context

// Obtén el valor de la cadena desde el contexto de la aplicación
val appName = getString(R.string.app_name)

//EJEMPLO ACCESO A UN COLOR
// Importa el paquete necesario si aún no lo has hecho
import android.content.Context
import androidx.core.content.ContextCompat

// Obtén una referencia al color
val color = ContextCompat.getColor(context, R.color.my_color)
```


5. RECURSOS ALTERNATIVOS

Como ya hemos comentado con anterioridad, una aplicación Android va a poder ser ejecutada en una gran variedad de dispositivos. El tamaño de pantalla, la resolución o el tipo de entradas puede variar mucho de un dispositivo a otro. Por otra parte, nuestra aplicación ha de estar preparada para diferentes modos de funcionamiento, como el modo automóvil o el modo noche, y para poder ejecutarse en diferentes idiomas.

A la hora de crear la interfaz de usuario hemos de tener en cuenta todas estas circunstancias. Afortunadamente, la plataforma Android nos proporciona una herramienta de gran potencia para resolver este problema, el uso de los recursos alternativos.

5.1. Diferentes layouts (horizontal y vertical)

Para probar esto vamos a crear una aplicación que nos va a permitir probar lo que deseamos:

1. Creamos una aplicación denominada **Tema4App1**.
2. Vamos a diseñar una pantalla que va a disponer de **un título y 5 botones** (vamos a emular en esta pantalla la pantalla inicial de un video juego).



3. El código de dicha pantalla es el siguiente:

```
<?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:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="VideoJuego"
        android:layout_marginTop="30dp"
        android:textSize="35sp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"/>

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:padding="30dp"
        android:layout_marginTop="60dp"
        app:layout_constraintTop_toTopOf="@id/editText"
        app:layout_constraintStart_toStartOf="parent">

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Iniciar juego"
            android:id="@+id/btnStart"
            android:layout_gravity="center_horizontal"
            android:textSize="25sp" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Configurar juego"
            android:id="@+id/btnConfig"
            android:textSize="25sp" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```

        android:text="Ver Ranking"
        android:id="@+id/btnRanking"
        android:textSize="25sp" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Acerca de"
    android:id="@+id/btnAbout"
    android:textSize="25sp" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Salir"
    android:id="@+id/btnExit"
    android:textSize="25sp" />
</LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>

```

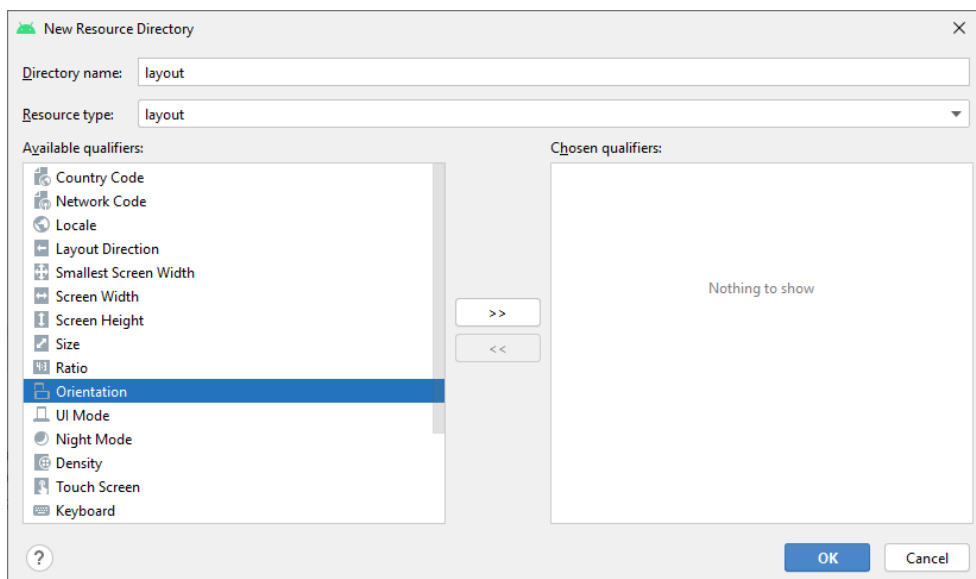
4. Los teléfonos móviles basados en Android permiten cambiar la configuración en **apaisado (landscape)** y **en vertical (portrait)**. Es posible que al usar un dispositivo pequeño se observe que el resultado de la vista que acabas de diseñar en vertical no queda del todo lo bien que desearíamos en apaisado.



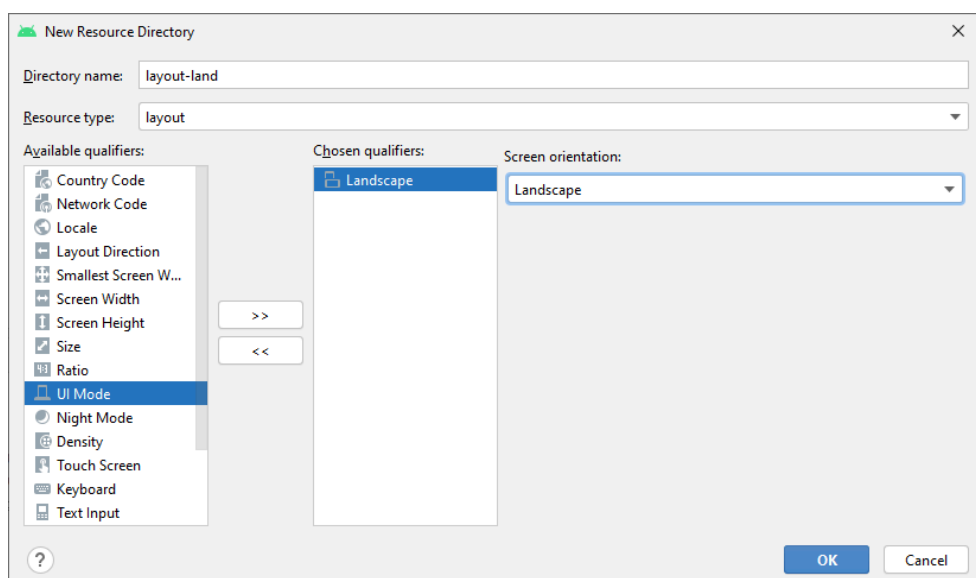
Si nos damos cuenta, el botón de Salir no se muestra en la pantalla. Para resolver este problema Android te permite diseñar una vista diferente para la configuración horizontal y

otra para vertical.

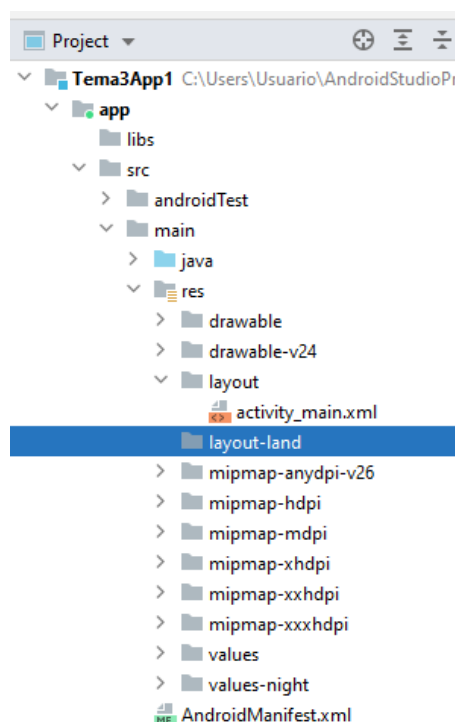
5. Crea la carpeta **/res/layout-land**. Para ello puedes pulsar con el botón derecho sobre la carpeta **/res** y selecciona **New > Android Resource Directory**.
6. Nos aparecerá la pantalla de **"New Resource Directory"**, donde seleccionaremos, en la opción **"Resource type"** la opción **layout**.



7. Ahora, en el apartado de **"Available qualifiers"** seleccionaremos la opción **orientation**.
8. Tras seleccionarlo podremos elegir la orientación de la pantalla. En la opción **"Screen orientation"** seleccionamos **Landscape**. Pulsamos el botón OK.



9. Ahora tendremos una nueva carpeta denominada **/res/layout-land**. Desde la visualización de Android no la veremos, por lo que **cambiaremos a la visualización de Project**.



10. Copia en ella el fichero **activity_main.xml** que se encuentra en la carpeta **/res/layout**. Para ello selecciona el fichero y pulsa Ctrl-C. Selecciona la carpeta destino (que será **res/layout-land**) y pulsa Ctrl-V.
11. **Crea una nueva vista** similar a la que ves a continuación: formada por un **LinearLayout** que contiene un **TextView** y un **TableLayout** con dos **Button** por columna.



NOTA:

Para

conseguir en un **TableLayout**, que las columnas se ajusten a todo el ancho de la tabla poner

stretchColumns="*". stretchColumns="0" significa que asigne el ancho sobrante a la primera columna. stretchColumns="1" significa que asigne el ancho sobrante a la segunda columna. stretchColumns="*" significa que asigne el ancho sobrante entre todas las columnas.

El código XML obtenido puede ser el siguiente:

```
<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">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/editText"
            android:gravity="center"
            android:text="VideoJuego"
            android:layout_marginTop="30dp"
            android:textSize="35sp" />

        <TableLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:stretchColumns="*">

            <TableRow
                android:layout_width="match_parent"
                android:layout_height="match_parent">

                <Button
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:text="Comenzar juego"
                    android:id="@+id/btnStart"
                    android:textSize="25sp" />

                <Button
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
```

```
        android:text="Configurar juego"
        android:id="@+id/btnConfig"
        android:textSize="25sp" />
    </TableRow>

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <Button
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="Ver ranking"
            android:id="@+id/btnRanking"
            android:textSize="25sp" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="Acerca de"
            android:id="@+id/btnAbout"
            android:textSize="25sp" />
    </TableRow>

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center">

        <Button
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="Salir"
            android:id="@+id/btnExit"
            android:textSize="25sp" />
    </TableRow>

</TableLayout>

</LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Android utiliza una lista de sufijos para expresar recursos alternativos. Estos sufijos pueden hacer referencia a la orientación del dispositivo, al lenguaje, la región, la densidad de píxeles, la resolución, el método de entrada...

El uso de recursos externos permite:

- Desacoplar código y recursos.
- Que el programa hará servir uno u otro recurso de acuerdo a la configuración del dispositivo sin tener que modificar ni una sola línea de código, simplemente creando las carpetas de los recursos adecuados.

Para cualquier tipo de recurso, puedes especificar cuáles van a ser los recursos usados por defecto y los recursos alternativos:

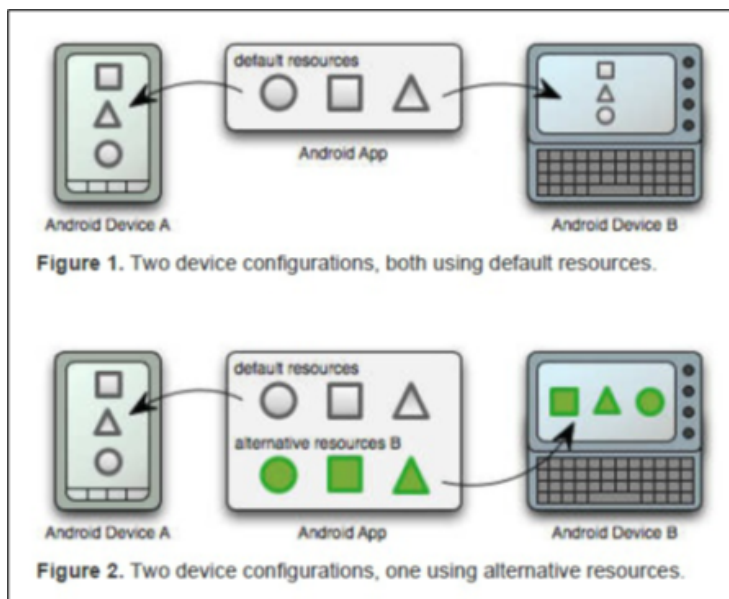
- Los recursos usados por defecto son aquellos que deberían ser usados sin considerar la configuración específica del dispositivo o cuando no hay recursos alternativos que coincidan con la configuración actual.
- Los recursos alternativos son aquellos que has diseñado para que sean usados con una configuración específica.

Android aplicará automáticamente los recursos apropiados relacionando la configuración actual del dispositivo con los nombres de las carpetas de recursos.

En las siguientes imágenes puedes ver cómo el sistema aplica el layout a dos dispositivos diferentes. En el primer caso, se muestra la situación en la que no hay recursos alternativos disponibles, por lo que se aplica el mismo layout. Pero en el segundo caso hay un recurso de layout alternativo para pantallas grandes, por lo que se utilizan layouts diferentes.



Otra posibilidad de verlo es la siguiente:



Para especificar que un grupo de recursos están asociados a una configuración específica, debe añadir un sufijo al nombre de la carpeta relacionándola con una determinada configuración. Se debe seguir el siguiente formato para nombrar la carpeta:

```
/res/tipoRecurso-cualificador
```

Donde tipoRecurso hace referencia al nombre del recurso a utilizar (por ejemplo: drawable, layout, etc.), y cualificador hace referencia al sufijo con el nombre de la configuración a usar.

Por ejemplo, mientras la estructura de la interfaz de usuario (layout) está guardada en la carpeta, /res/layout, puedes especificar un layout diferente para ser usado cuando la pantalla esté en sentido apaisado, para lo cual debes guardarlo en la carpeta /res/layout-land.

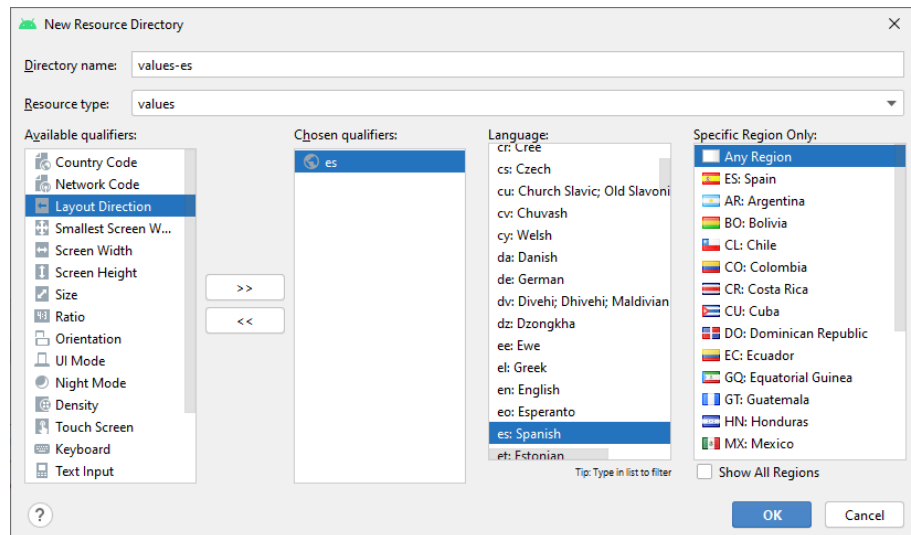
Por ejemplo, si queremos traducir nuestra aplicación al inglés, español y francés, siendo el primer idioma el usado por defecto, crearemos tres versiones del fichero strings.xml y lo guardaremos en los siguientes tres directorios:

```
/res/values/strings.xml  
/res/values-es/strings.xml  
/res/values-fr/strings.xml
```

5.2. Traducción de una aplicación

Vamos a realizar una traducción de la aplicación **Tema4App2**.

1. Crea la carpeta **/res/values-es** tal y como hemos creado antes la carpeta **res/layout-land**.
2. Selecciona **Locale** y después busca **es:Spanish** → **Any Region**.



3. Sustituye el código del fichero del **strings.xml** de la carpeta **/res/values** por el siguiente (este código va a crear las entradas correspondientes a los textos de la aplicación).

```
<resources>
    <string name="app_name">Tema4App1</string>
    <string name="game">Game</string>
    <string name="start">Start game</string>
    <string name="settings">Settings game</string>
    <string name="ranking">Ranking</string>
    <string name="about">About</string>
    <string name="exit">Exit</string>
    <string name="action_settings">Settings</string>
</resources>
```

4. Ahora nos vamos a **cada layout y sustituimos el texto** de cada elemento por la entrada en el string correspondiente. Por ejemplo, en el botón **start** quedará de la siguiente forma:

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

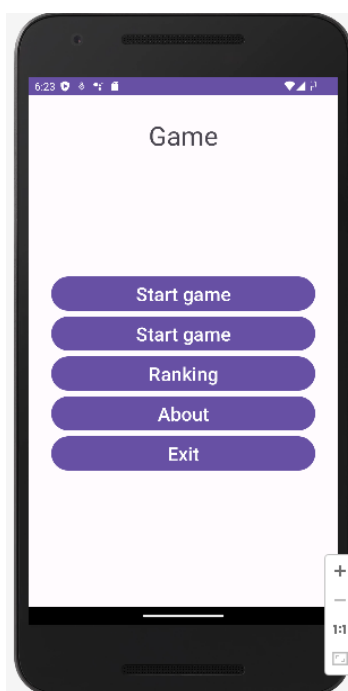
```
android:text="@string/start"  
android:id="@+id/btnStart"  
android:layout_gravity="center_horizontal"  
android:textSize="25dp" />
```

Fijaros como ha cambiado la opción `android:text`, y ahora hace referencia a la entrada `start` del fichero `strings.xml`. Recordar de cambiar el layout de `/res/layout` y de `/res/layout-land`.

5. **Copiar el fichero** `/res/values/strings.xml` a la carpeta `/res/values-es/strings.xml`.
6. **Traduce en este fichero** de la carpeta `/res/values-es` todas las cadenas al castellano (ojo, las cadenas, no los nombres de las variables `string`). El resultado debe el siguiente:

```
<resources>  
  <string name="app_name">Tema4App2</string>  
  <string name="game">Videojuego</string>  
  <string name="start">Iniciar juego</string>  
  <string name="settings">Configurar juego</string>  
  <string name="ranking">Ver Ranking</string>  
  <string name="about">Acerca de</string>  
  <string name="exit">Salir</string>  
  <string name="action_settings">Settings</string>  
</resources>
```

7. **Ejecuta la aplicación** y verás que aparecen todos los textos en inglés.



8. Vamos a **cambiar la configuración de idioma de la máquina virtual de Android**. Para ello, debemos acceder a los Ajustes del dispositivo (Settings) como si accedieras a nuestro móvil y selecciona la opción Mi dispositivo (Personal) > Idioma e introducción (Language & Input). Dentro de esta opción selecciona como idioma en la opción Lenguaje (Language) el idioma Español (España).

NOTA: Observa que en otros idiomas permite seleccionar tanto el idioma como la región. Por desgracia, para el español solo permite dos regiones: España y Estados Unidos.

9. Observa cómo se ha traducido el texto.



10. ¿Qué pasaría si nuestra aplicación se ejecuta en un terminal configurado en chino? ¿Aparecerán las cadenas en castellano o en inglés? ¿Piensas que esta es una buena decisión de diseño? ¿Cómo lo podrías solucionar?

5.3 Normas en la forma de nombrar recursos con sus sufijos

Resulta posible indicar varios sufijos concatenados de la siguiente forma:

```
/res/tipoRecurso-cualificador1-cualificador2-cualificador3-...
```

Por ejemplo:

```
/res/values-en-rUS/strings.xml  
/res/values-en-rUK/strings.xml
```

Pero cuidado, Android establece un orden a la hora de encadenar sufijos. Puedes encontrar una lista de estos sufijos en este enlace:

- <http://developer.android.com/guide/topics/resources/providing-resources.html>

Aun así, vamos a comentar los posibles nombres de cualificadores que se pueden usar, agrupados por categorías. Esta lista está ordenada por el orden como se han de indicar:

1. MCC y MNC:

- Código de país de móvil (mobile country code MCC) seguido opcionalmente del código de red de móvil (mobile network code) de la tarjeta SIM que tenga el dispositivo. Por ejemplo, `mcc214-mnc07` para Movistar, `mcc214-mnc01` para Vodafone o `mcc214-mnc03` para Orange. Consulta la página de Wikipedia para más códigos.

2. Idioma y región:

- El código de idioma está definido por 2 letras según ISO 639-1, y puede ir seguido opcionalmente del código de región ISO 3166-1-alpha-2 precedido del carácter 'r'. Por ejemplo, `"es"` para Español de cualquier región, o `"es-rES"` para español de España y `"es-rMX"` para español de México. `"en"` es inglés para cualquier región, `"en-rUS"` es inglés de Estados Unidos y `"en-rGB"` es inglés de Gran Bretaña.

3. Tamaño mínimo de pantalla:

- Indica el tamaño más pequeño de la pantalla (considerando tanto alto como ancho).

Esta medida es indiferente a la orientación de la pantalla. Se indica la medida precedida de "sw" y seguida de "dp", por ejemplo sw320dp o sw720dp. Desde API 13.

4. Ancho disponible:

- Especifica el ancho mínimo de pantalla disponible, cambiando su valor cuando la orientación de la pantalla varíe entre apaisado o vertical considerando en cada caso el ancho correspondiente. Se indica la medida precedida de "w" y seguida de "dp", por ejemplo w720dp o w1024dp. Desde API 13.

5. Altura disponible:

- Especifica la altura mínima de pantalla disponible de manera similar a la anterior. Por ejemplo: h720dp o h1024dp. Desde API 13.

6. Tamaño de pantalla:

- small: 320x426 dp aprox.
- normal: 320x470 dp aprox.
- large: 480x640 dp aprox.
- xlarge: 720x960 dp aprox. Desde API 9.
- Esta opción se encuentra ya en estado deprecated, por lo que es necesario utilizar sw. Se aconseja, no obligatoria, pero si muy recomendable, ya que aclara muchas cosas, visitar http://developer.android.com/guide/practices/screens_support.html.

7. Aspecto de la pantalla:

- long: Pantallas largas como WQVGA, WVGA y FWVGA.
- notlong: Pantallas menos largas como QVGA, HVGA y VGA.

8. Orientación de la pantalla:

- port: (Portrait) Modo retrato (vertical).
- land: (Landscape) Modo apaisado (horizontal).

9. Modo de interfaz de usuario (desde API 8):

- car: Dispositivo conectado a un soporte (dock) de coche.
- desk: Dispositivo conectado a un soporte de escritorio.
- televisión: Dispositivo mostrado en una televisión. Desde API 13.
- appliance: Dispositivo actuando como appliance (diseñado para actuar para un propósito específico), sin pantalla.

10. Modo nocturno (desde API 8):

- night: En horario nocturno.
- notnight: En horario diurno.

11. Densidad de puntos de pantalla(dpi):

- ldpi: 120dpi aprox.
- mdpi: 160dpi aprox.
- hdpi: 240dpi aprox.
- xhdpi: 320dpi aprox. (desde API 8)
- nodpi: Se puede usar para imágenes que no se desea escalar en función de la densidad de pantalla.
- tvdpi: Entre mdpi y hdpi. 213dpi aprox. (desde API 13)
- Hay un ratio de escala de 3:4:6:8 entre los cuatro primeros tipos de densidad. Así, una imagen de 9x9 en ldpi es de 12x12 en mdpi, 18x18 en hdpi y 24x24 en xhdpi.

12. Tipo de pantalla táctil:

- notouch: No tiene pantalla táctil.
- finger: Tiene pantalla táctil.

13. Disponibilidad de teclado:

- keysexposed: El dispositivo tiene un teclado físico disponible.
- keyshidden: Tiene un teclado físico, pero está oculto.
- keysoft: Tiene un teclado virtual (por software).

14. Tipo de teclado físico:

- nokeys: El dispositivo no tiene teclas físicas.
- qwerty: Tiene un teclado físico tipo qwerty.
- 12key: Tiene un teclado físico de 12 teclas.

15. Disponibilidad de teclas de navegación:

- navexposed: Las teclas de navegación están disponibles.
- navhidden: Las teclas de navegación no están disponibles.

16. Método principal de navegación no táctil:

- nonav: El dispositivo no tiene otro medio de navegación que no sea la pantalla táctil.
- dpad: Tiene un pad (teclado) direccional.
- trackball: Tiene un trackball (bola).

- wheel: Tiene ruedas para la navegación (poco común).

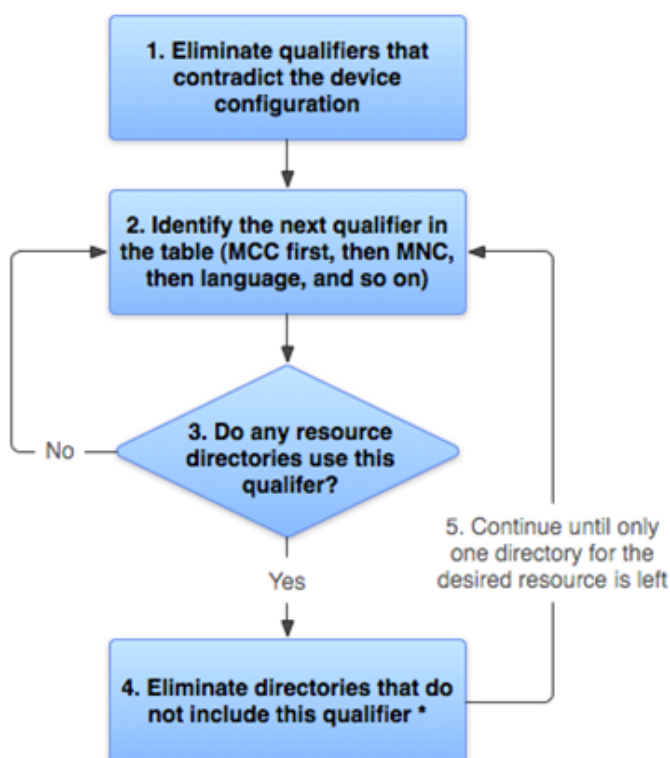
17. Versión de la plataforma:

- Nivel de API soportado por el dispositivo. Por ejemplo, v3, v4, v7, etc.

Los nombres de cualificadores deben seguir además unas reglas:

- Se pueden especificar varios cualificadores a un mismo conjunto de recursos, separándolos por guiones. Por ejemplo: drawable-en-rUS-land se aplicaría a dispositivos configurados con el idioma inglés de Estados Unidos y cuando esté en modo apaisado.
- Los cualificadores deben estar en el orden de la lista anterior. Por ejemplo:
 - Incorrecto: drawable-hdpi-port.
 - Correcto: drawable-port-hdpi.
- Las carpetas de recursos alternativos no pueden ser anidadas. Por ejemplo:
 - Incorrecto: /res/drawable/drawable-en
 - Correcto: /res/drawable-en
- Los nombres de cualificadores no consideran las diferencias entre mayúsculas y minúsculas.
- Solo es válido un valor para cada tipo de cualificador. Por ejemplo:
 - Incorrecto: drawable-es-fr
 - Correcto: drawable-es y otra carpeta drawable-fr.
- Lo que sí se puede hacer es crear alias entre recursos.

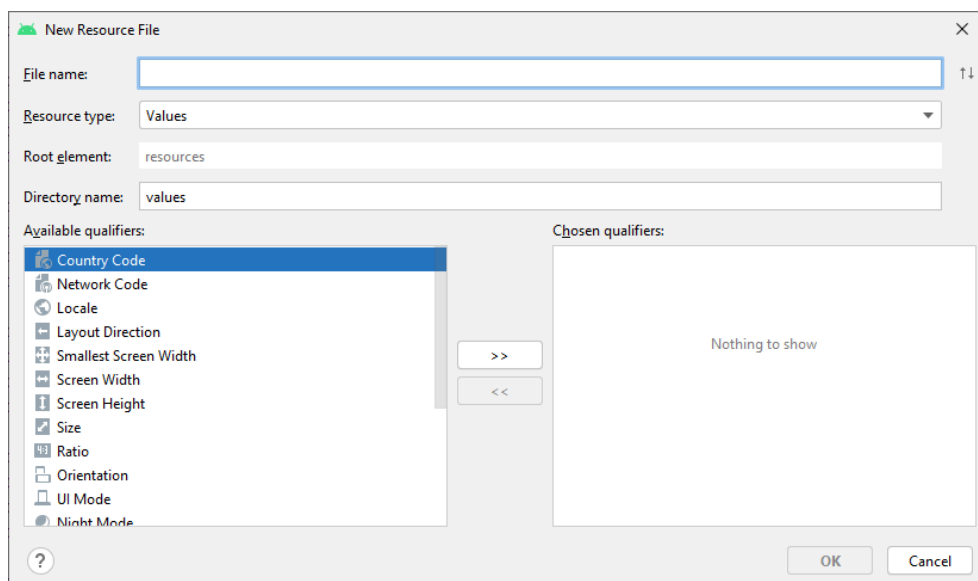
Android utiliza el siguiente algoritmo para seleccionar los recursos:



* If the qualifier is screen density, the system selects the "best match" and the process is done

Android utilizará automáticamente en la aplicación los recursos basados en la configuración actual del dispositivo. Cada vez que se solicita un recurso, Android comprueba si hay carpetas de recursos alternativos que contengan el recurso solicitado y entonces localiza el recurso que mejor se adapta a la configuración actual. Si no hay recursos alternativos para la configuración actual, Android usará los recursos correspondientes por defecto (los que no incluyen un cualificador).

Para ver los sufijos disponibles, también puedes pulsar con el botón derecho sobre la carpeta /res/ y seleccionar Android Resource File. Esta opción te permite crear un nuevo fichero XML y poner el sufijo deseado de forma y orden correcto, a través del asistente.



New Resource File

File name: 14

Resource type:

Root element:

Directory name:

Available qualifiers:

- ☒ Country Code
- ☐ Network Code
- ☐ Locale
- ☐ Layout Direction
- ☐ Smallest Screen Width
- ☐ Screen Width
- ☐ Screen Height
- ☒ Size
- ☐ Ratio
- ☐ Orientation
- ☐ UI Mode
- ☐ Night Mode

Chosen qualifiers:

Nothing to show

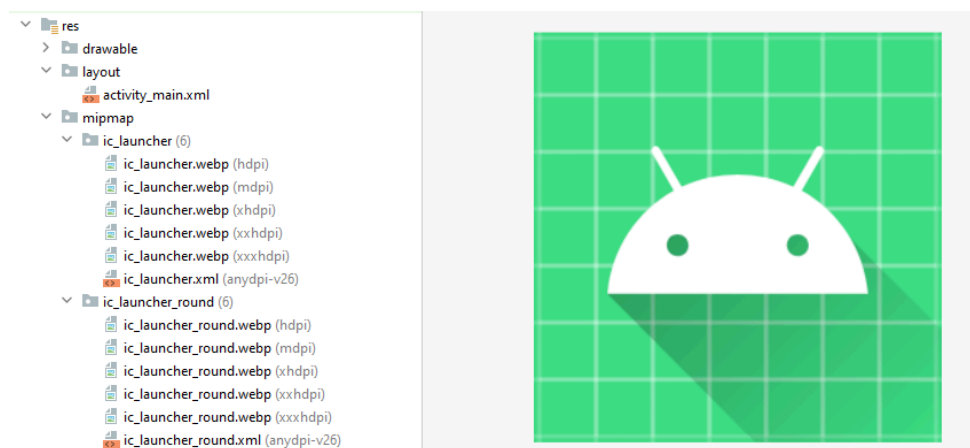
>> <<

OK Cancel

5.4. Icono para lanzar la aplicación

Otro ejemplo de utilización de recursos diferenciados lo podemos ver con el icono que se utiliza para lanzar la aplicación. Hasta ahora, hemos usado como **ic_launcher** (icono de la App) el que nos ha dado Android Studio por defecto, concretamente la cara del robot de Android.

Observa cómo, al crear una aplicación, este icono se crea en seis carpetas diferentes, para densidades distintas de pantalla. Android carga automáticamente el correcto según la pantalla del dispositivo. Recuerda que el icono lanzador se define en el Manifest por medio del atributo **android:icon="@mipmap/ic_launcher"**.

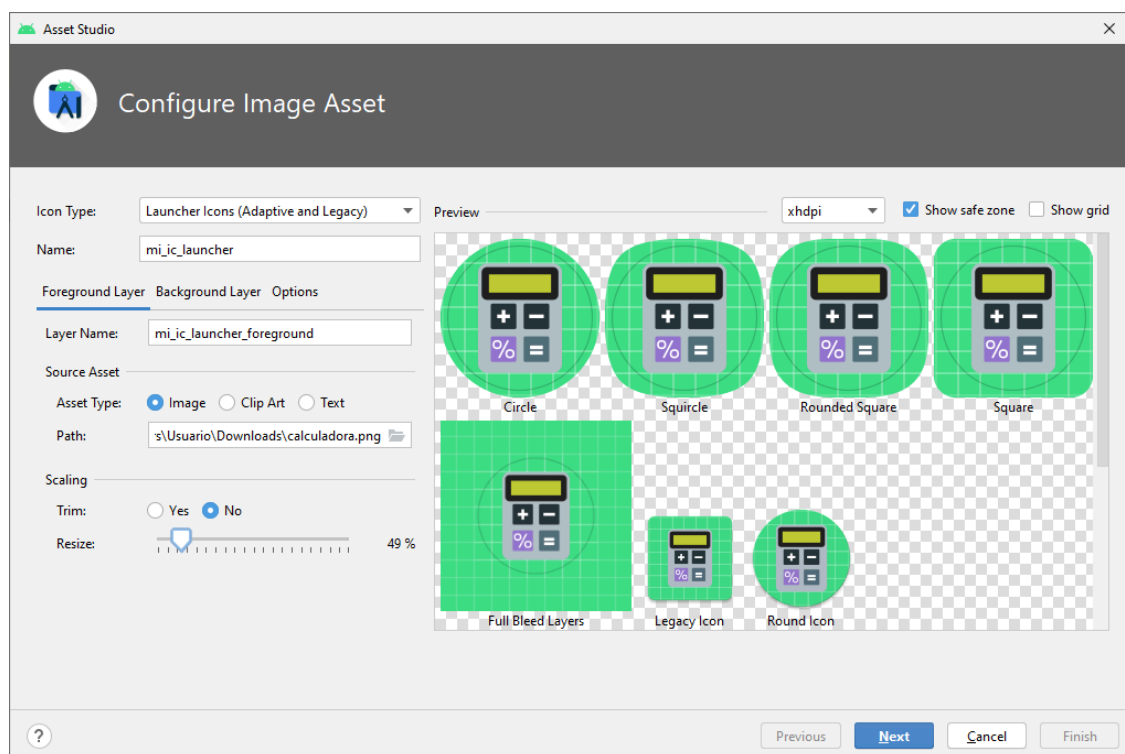


Como podemos observar, además del icono cuadrado hay otra versión **round (ic_launcher_round)**, para móviles con estilos redondeados en los iconos. Además, a partir de la **API Android v26** se puede usar un icono de tamaño adaptativo definido mediante **XML**.

5.4.1. Generar icono con todas las densidades

Vamos a ver cómo generar un icono con todas las densidades. Es evidente que necesitaremos una imagen base cuya calidad mínima sea suficiente para la densidad máxima, y generar las otras imágenes con menos requerimientos. Se podría usar como base una imagen diseñada con cualquier buen editor fotográfico.

1. Partiremos de un icono buscado en la web de una resolución de 1024x1024 píxeles para generar las demás. En mi caso, he buscado uno en la siguiente página web:
 - <https://www.iconfinder.com>
2. Vamos a la carpeta **/res/mipmap** y botón derecho **New > Image Asset**.
3. Veremos el aspecto de **Android Asset Studio**.



Tal y como se muestra en la imagen anterior escogemos:

- **Icon Type** → Launcher Icons (Adaptive and Legacy)
- **Name** → mi_ic_launcher
- **Asset Type** → Image
- **Path** → buscamos el archivo que nos hemos bajado de la web (en formato .png)
- **Trim** → No
- **Resize** → 49%.

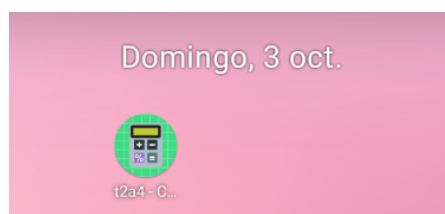
4. Le damos a **Next** y nos pedirá confirmación, repasamos todo y **Finish**.

Ahora veremos el nuevo icono **mi_ic_launcher** en el árbol, en tres carpetas distintas (normal, foreground y round), con sus cinco densidades y el xml adaptativo.

Para usarlo deberíamos cambiar en el **AndroidManifest.xml** los atributos icon y roundIcon por:

```
android:icon="@mipmap/mi_ic_launcher"  
android:roundIcon="@mipmap/mi_ic_launcher_round"
```

Pruebo a instalar, y podemos observar en la pantalla de inicio la diferencia del antes y el ahora en cuanto al icono se refiere (según tu móvil te saldrá redondo o cuadrado).



Si quieres ya se pueden borrar los viejos iconos ic_launcher, desde el mismo árbol del proyecto (botón derecho y delete).

6. RECURSOS DEL SISTEMA

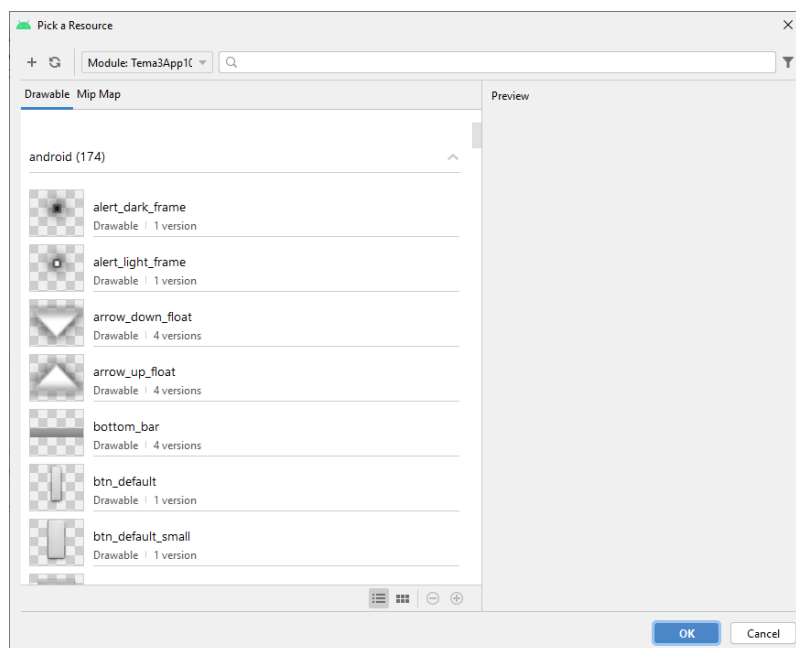
Además de los recursos que podamos añadir a nuestra aplicación, también podemos utilizar una serie de recursos que han sido incluidos en el sistema.

Usar recursos del sistema tiene muchas ventajas. No consumen memoria en nuestra aplicación, al estar ya incorporados en el sistema. Además, los usuarios están familiarizados con ellos. Por ejemplo, si utilizamos el recurso `android.R.drawable.ic_menu_edit` se mostrará al usuario el icono:



. Muy posiblemente el usuario ya está familiarizado con este icono y lo asocie a la acción de editar. Otra ventaja es que los recursos del sistema se adaptan a las diferentes versiones de Android. Si se utiliza el tema `android.R.style.Theme_Panel` este es bastante diferente en cada una de las versiones, pero seguro que estará en consonancia con el resto de estilos para esta versión. Lo mismo ocurre con el icono anterior. Este icono es diferente en algunas versiones, pero al usar un recurso del sistema nos aseguramos que se mostrará el adecuado a la versión del usuario. Finalmente, estos recursos se adaptan siempre a las configuraciones locales. Si yo utilizo el recurso `android.R.string.cancel` este será “Cancelar”, “Cancel”, “取消”,... según el idioma escogido por el usuario.

Para utilizar los recursos del sistema, añadiremos a la interfaz el elemento que queramos visualizar, por ejemplo, una `ImageView` y automáticamente nos aparecerá la interfaz para seleccionar el recurso:



Así que, dentro del apartado android encontraremos todos los iconos que pertenecen al sistema, de hecho, una vez selecciones uno de ellos observamos cómo está referenciado:

```
@android:drawable/ic_lock_lock
```

7. ESTILOS Y TEMAS

Si tienes experiencia con el diseño de páginas Web habrás advertido grandes similitudes entre HTML y el diseño de layouts. En los dos casos se utiliza un lenguaje de marcado y se trata de crear diseños independientes del tamaño de la pantalla donde serán visualizados. En el diseño web resulta clave las hojas de estilo en cascada (CSS) que permiten crear un patrón de diseño y aplicarlo a varias páginas. **Cuando diseñes los layouts de tu aplicación vas a poder utilizar unas herramientas similares conocidas como estilos y temas. Te permitirán crear patrones de estilo que podrán ser utilizados en cualquier parte de la aplicación.** Estas herramientas te ahorrarán mucho trabajo y te permitirán conseguir un diseño homogéneo en toda tu aplicación.

Para profundizar más sobre esta temática y poder ver las últimas novedades en tema de diseño de estilo y temas, se recomienda la siguiente lectura:

- <https://developer.android.com/guide/topics/ui/look-and-feel/themes>

7.1. Estilos

Un estilo es una colección de propiedades que definen el formato y apariencia que tendrá una vista (un componente visual). Podemos especificar cosas como tamaño, márgenes, color, fuentes, etc. Un estilo se define en ficheros XML, diferente al fichero XML layout que lo utiliza.

Veamos un ejemplo. El siguiente código:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="Un texto" />
```

Es equivalente a escribir:

```
<TextView
    style="@style/MiEstilo"
    android:text="Un texto" />
```


En el fichero themes.xml escribiremos el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="MiEstilo" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Observa como un estilo puede heredar todas las propiedades de un padre (parámetro parent) y a partir de estas propiedades realizar modificaciones.

7.2. Heredar de un estilo propio

Si vas a heredar de un estilo definido por ti, no es necesario utilizar el atributo parent. Por el contrario, puedes utilizar el mismo nombre de un estilo ya creado y completar el nombre con un punto más un sufijo. Por ejemplo:

```
<style name="MiEstilo.grande">
    <item name="android:textSize">18pt</item>
</style>
```

Crearía un nuevo estilo que sería igual a MiEstilo más la nueva propiedad indicada. A su vez puedes definir otro estilo a partir de este:

```
<style name="MiEstilo.grande.negrita">
    <item name="android:textStyle">bold</item>
</style>
```

Vamos a crear un estilo:

1. Creamos un proyecto denominado **Tema4App3** y copiamos el layout principal del diseño vertical y el archivo strings.xml generados en el proyecto Tema4App1 a este.
2. Crea un **nuevo estilo** con nombre **EstiloTexto**, que descienda de:

`@android:style/TextAppearance`

3. **Aplicalo el estilo al título** que aparece en el layout `/res/layout/activity_main.xml`.
4. Crea un **nuevo estilo** con el nombre **EstiloTexto.Botones**. Este ha de modificar el color del texto, indicando que el nuevo color sea `#FF0000`, y que el `layout_width` sea `wrap_content`.
5. **Aplicalo a todos los botones** del layout.
6. **Visualiza el resultado**, que sería como esto:



Solució para el ejemplo anterior:

Una solució para el texto sería la siguiente:

```
<style name="EstiloTexto" parent="@android:style/TextAppearance">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#191970</item>
    <item name="android:typeface">monospace</item>
</style>
```

Y el código solución para los botones sería:

```
<style name="EstiloTexto.Botones">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:textColor">#FF0000</item>
</style>
```

El código fuente del Layout /res/layout/activity_main.xml sería el siguiente:

```
<?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:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/game"
        android:layout_marginTop="30dp"
        android:textSize="35sp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        style="@style/EstiloTexto"/>

    <LinearLayout
```

```
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:padding="30dp"
android:layout_marginTop="60dp"
app:layout_constraintTop_toTopOf="@id/editText"
app:layout_constraintStart_toStartOf="parent">

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/start"
    android:id="@+id/btnStart"
    android:layout_gravity="center_horizontal"
    android:textSize="25sp"
    style="@style/EstiloTexto.Botones"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/start"
    android:id="@+id/btnConfig"
    android:textSize="25sp"
    style="@style/EstiloTexto.Botones"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/ranking"
    android:id="@+id/btnRanking"
    android:textSize="25sp"
    style="@style/EstiloTexto.Botones"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/about"
    android:id="@+id/btnAbout"
    android:textSize="25sp"
    style="@style/EstiloTexto.Botones"/>

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/exit"
    android:id="@+id/btnExit"
    android:textSize="25sp"
    style="@style/EstiloTexto.Botones"/>
</LinearLayout></androidx.constraintlayout.widget.ConstraintLayout>
```

7.3. Los temas

Un tema es un estilo aplicado a toda una actividad o aplicación, en lugar de a una vista individual. Cada elemento del estilo solo se aplicará a aquellos elementos donde sea posible. Por ejemplo, CodeFont solo afectará al texto.

Para aplicar un tema a toda una aplicación edita el fichero AndroidManifest.xml y añade el parámetro android:theme en la etiqueta application:

```
<application
    android:theme="@style/MiTema">
```

También puedes aplicar un tema a una actividad en concreto:

```
<activity
    android:theme="@style/MiTema">
```

La primera personalización que puedes hacer es personalizar el tema modificando la paleta de colores, para ello accede al fichero colors y cambia los colores, para ayudarte puedes utilizar el “Palette generator”:

- <https://m2.material.io/design/material-theming/overview.html#material-theming>

Además de crear tus propios temas vas a poder utilizar algunos disponibles en el sistema.

Vamos a aplicar un tema del sistema a una de las aplicaciones con las que trabajamos:

1. Crea un proyecto denominado **Tema4App4**.
2. Aplica a la actividad principal el tema (AndroidManifest.xml):

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

3. Visualiza el resultado.



8. ESCUCHADORES DE EVENTOS EN ANDROID

Android captura los eventos que genera la interacción con el usuario de dos formas distintas.

Un escuchador de eventos o event listener no es más que una interfaz que responde a las distintas interacciones con el usuario, llamando a una serie de métodos predefinidos.

Vamos a ver algunas de las interfaces más usadas y los métodos que contienen:

- **View.OnClickListener:** Con el método `onClick`, que será llamado cuando el usuario toque una vista. Por ejemplo, un botón.
- **View.OnLongClickListener:** Con el método `onLongClick`, que será llamado cuando el usuario haga una pulsación larga sobre una vista. Podemos usarlo por ejemplo para sacar un menú al mantener un ítem en una lista.
- **View.OnFocusChangeListener:** con el método `onFocusChange`, que será llamado cuando cambie el foco de la vista. El foco significa que es la vista actual seleccionada, con la que podemos interactuar. Por ejemplo, cuando pulsamos encima de un `EditText` y sale el cursor para escribir, este `EditText` tiene el foco. Más tarde, cuando por ejemplo cambiemos a otro `EditText` o pulsemos un botón, es anterior, pierde el foco. En los 2 casos, tanto cuando obtiene el foco como cuando lo pierde, se ejecutará el método `onFocusChange`.
- **View.OnKeyListener:** con el método `onKey`, que será llamado al pulsar una tecla.

El sistema nos avisará de esta interacción, SOLO si registramos un escuchador de eventos para esa vista con su método asociado. Por ejemplo, para el evento `onClick`, se registra llamando a `setOnClickListener()`, pasando como argumento la clase que implemente su interfaz. Es aconsejable utilizar este sistema.

Un ejemplo para implementar esto en Kotlin sería el que se muestra a continuación:

```
val miBoton = findViewById<Button>(R.id.miBoton)

miBoton.setOnClickListener {
    // Código a ejecutar cuando se haga clic en el botón
    Toast.makeText(this, "Clic en el botón",
        Toast.LENGTH_SHORT).show()
}
```

```
}
```

Y a continuación se muestra un ejemplo con otro tipo de evento:

```
val miEditText = findViewById<EditText>(R.id.miEditText)

miEditText.setOnFocusChangeListener { view, hasFocus ->
    if (hasFocus) {
        // Código a ejecutar cuando el EditText gana el foco
        Toast.makeText(this, "EditText ha ganado el foco",
            Toast.LENGTH_SHORT).show()
    } else {
        // Código a ejecutar cuando el EditText pierde el foco
        Toast.makeText(this, "EditText ha perdido el foco",
            Toast.LENGTH_SHORT).show()
    }
}
```


9. ANIMACIONES LOTTIE EN APLICACIONES ANDROID

Hay diferentes formas de realizar animaciones en Android, pero vamos a utilizar las animaciones de tipo Lottie. Para ello, podemos ver un ejemplo completo en el blog de la misma web:

- <https://lottiefiles.com/blog/working-with-lottie/getting-started-with-lottie-animations-in-an-droid-app>

