

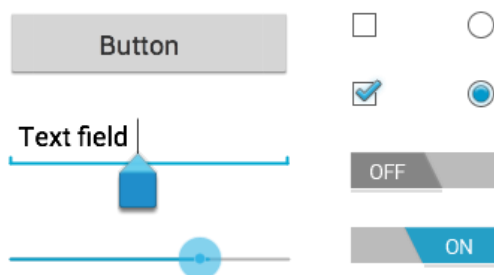
## INDICE

---

<b>1.1. CONTROLES</b>	<b>1</b>
<b>1.2. TEXTVIEW. DEFINICIÓN DE RECURSOS XML</b>	<b>2</b>
<b>1.2.1. INTRODUCCIÓN</b>	<b>2</b>
<b>1.2.2. CASOS PRÁCTICOS</b>	<b>4</b>
<b>1.2.3. ACCEDER Y MANIPULAR EL CONTROL DESDE JAVA</b>	<b>6</b>
<b>1.2.4. MANIPULACIÓN HTML DE UNA ETIQUETA DE TEXTO</b>	<b>9</b>
<b>1.2.5. DEFINICIÓN DE CONSTANTES/RECURSOS XML</b>	<b>14</b>
<b>1.2.6. CLICKABLE / NO CLICKABLE</b>	<b>22</b>
<b>1.3. ACCESIBILIDAD</b>	<b>23</b>
<b>1.4. EMPLEANDO RECURSOS DEFINIDOS RES</b>	<b>25</b>
<b>1.5. ATRIBUTOS QUE DEBES CONOCER</b>	<b>27</b>
<b>1.6. MÉTODOS MÍNIMOS QUE DEBES CONOCER EN EL MANEJO DE TEXTVIEWS</b>	<b>28</b>

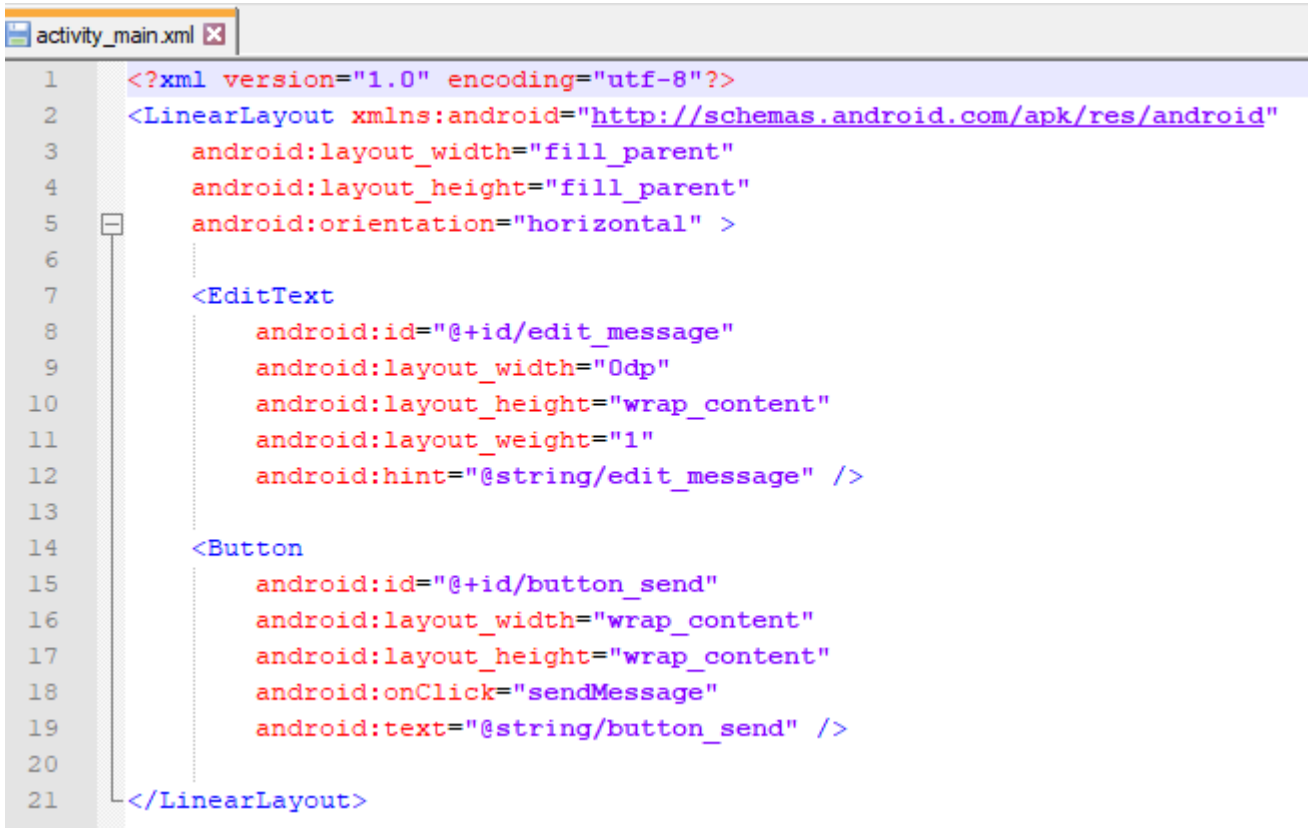
## 1.1. Controles

- Los controles de la UI de usuario son los elementos interactivos de las aplicaciones.
- La siguiente imagen muestra un ejemplo de los más utilizados:



- Todos los componentes visuales están en el paquete: **android.widget** (consulte las referencias a continuación)
- Se puede agregar un control a un layout a través del componente XML o mediante Java en tiempo de ejecución.
- En las siguientes secciones usaremos los controles más comunes.
- Además de definir los controles en XML también veremos una manipulación básica de ellos en Java.
- Además de explicar un control, en algunos casos se va a aprovechar para explicar otros conceptos.
- Corresponde al usuario estudiar aquellos controles en los que está interesado en el paquete anterior y no están contemplados en el tema.

- El siguiente es un ejemplo de un diseño XML con una etiqueta (*EditText*) y un botón:



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      android:orientation="horizontal" >
6
7      <EditText
8          android:id="@+id/edit_message"
9          android:layout_width="0dp"
10         android:layout_height="wrap_content"
11         android:layout_weight="1"
12         android:hint="@string/edit_message" />
13
14     <Button
15         android:id="@+id/button_send"
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:onClick="sendMessage"
19         android:text="@string/button_send" />
20
21 </LinearLayout>
```

➤ Referencias:

- ✓ Paquete android.widget: <https://developer.android.com/reference/android/widget/package-summary.html>

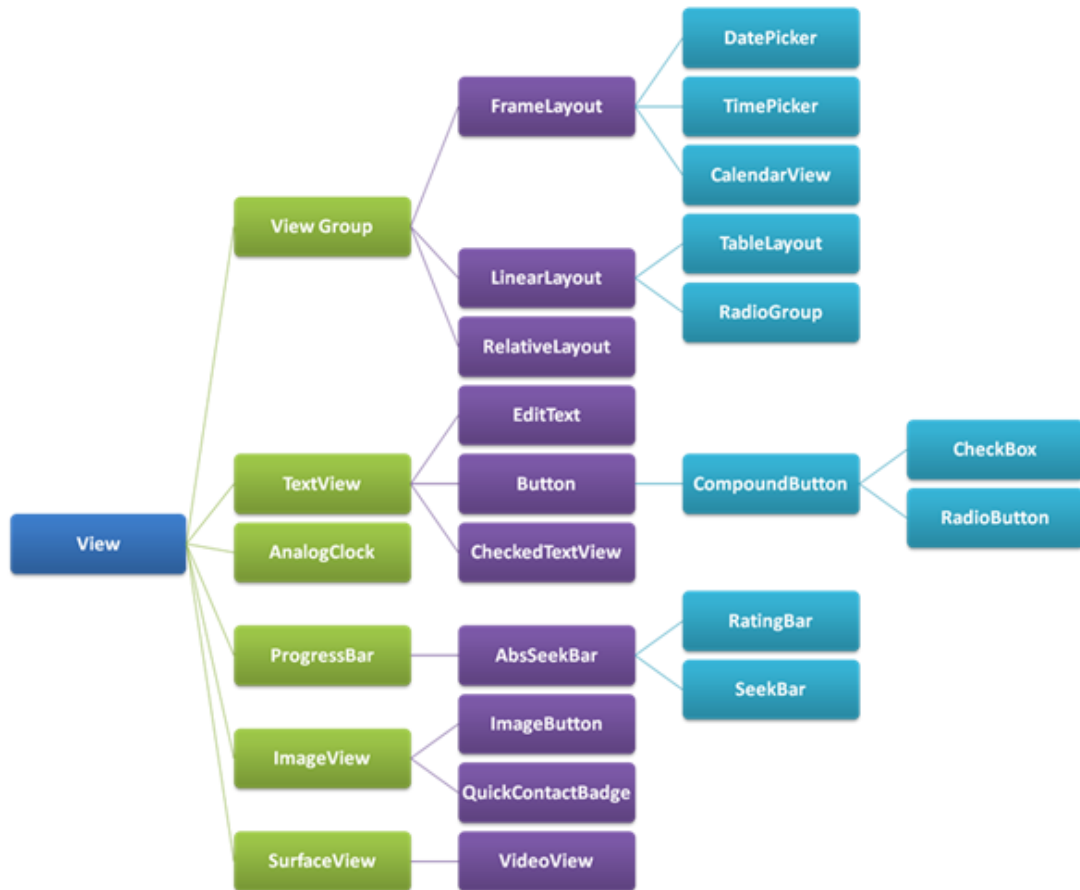
## 1.2. TextView. Definición de recursos XML

### 1.2.1.Introducción

- Un **TextView** es una etiqueta de texto que se utiliza para mostrar texto en la aplicación.
- Este control es una subclase de la clase **View**.
- Opcionalmente, puede permitir la edición del contenido.
- La clase que implementa el control está configurada para no permitir la edición.
- Aprovechando que este es el primer control que veremos, también explicaremos cómo se definen las constantes para su uso en recursos XML.

## ➤ Referencias:

- ✓ La clase **View**: <https://developer.android.com/reference/android/view/View.html>
  - Antes de continuar, es importante familiarizarse (no es necesario aprender nada) con los métodos, subclases y atributos que tiene esta clase a los que se aferrarán todos los demás controles. Con lo cual, además de detenerse en la siguiente imagen, es recomendable hacer clic en el enlace de arriba.



Classes from Android View Hierarchy

- Control **TextView**: <https://developer.android.com/reference/android/widget/TextView.html>
  - 🔗 Observar en el enlace anterior el valor del atributo editable.
- Recomendaciones de tipografía: <https://material.io/design/typography/the-type-system.html#type-scale>
  - 🔗 Observar en el enlace de arriba el tamaño recomendado en px del lanzador de una aplicación en Google Play.
- Colores: <https://developer.android.com/guide/topics/resources/more-resources.html#Color>
  - 🔗 Observar en qué formatos se puede describir un color.

## 1.2.2.Casos prácticos

- Suponemos que ya hemos creado el proyecto inicial como se tema 2.3.

Si no lo hemos creado antes, cree un paquete llamado **UI** como un subpaquete de su paquete principal.

Dentro del paquete de UI, crearemos un nuevo paquete llamado: **CajaTexto**.

- Dentro del paquete **CajaTexto** crea una nueva 'Empty Activity' llamada: **UD02\_01\_TextView** tipo Launcher y sin compatibilidad.

Modificar el archivo **AndroidManifest.xml** y agregar un label a la activity como vimos al crear el proyecto base .

- Comencemos creando un diseño con 2 TextViews, donde el segundo TextView cambia en tiempo de ejecución (cuando se inicia la aplicación).

Nota: El ejemplo usa un LinearLayout pero puede usar cualquiera de los diseños que se ven arriba.



- La imagen muestra los 2 TextViews en tiempo de diseño.
- Leer el contenido de la etiqueta azul.
- El código XML del layout asociado con esa imagen es:

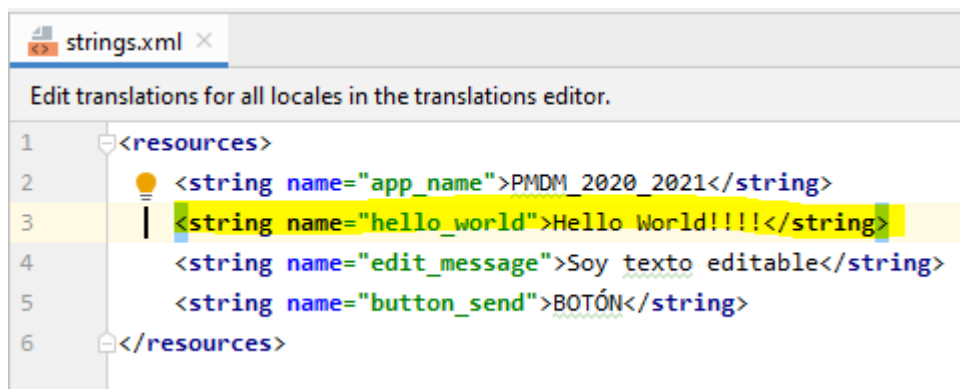


```

1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:orientation="vertical"
8      android:padding="20sp"
9      tools:context=".UI.CajaTexto.UD02_01_TextView">
10
11     <TextView
12         android:id="@+id/txtOriginal_UD02_01_TextView"
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:layout_marginEnd="15dp"
16         android:text="@string/hello_world" />
17
18     <TextView
19         android:id="@+id/txtJava_UD02_01_TextView"
20         android:layout_width="wrap_content"
21         android:layout_height="wrap_content"
22         android:textColor="#00F"
23         android:text="Da igual lo que escribamos aquí. En este ejemplo,
24                     al lanzar la aplicación, se va a cambiar el texto" />
25 </LinearLayout>

```

- En las líneas 12 y 19 se asocia un ID a cada control TextView, que luego se utilizará en Java.
- Observar las diferencias entre las líneas 16 y 23. La primera mostrará el texto que contiene la constante definida en XML (en otro archivo XML), la segunda muestra el texto directamente.
- Debe tener la constante **@string/hello\_world** definida en el archivo de recursos: **/res/values/strings.xml**, como se muestra después de la imagen.



```

1  <resources>
2      <string name="app_name">PMDM_2020_2021</string>
3      <string name="hello_world">Hello World!!!!</string>
4      <string name="edit_message">Soy texto editable</string>
5      <string name="button_send">BOTÓN</string>
6  </resources>

```

- Como ya se ha mencionado, es recomendable utilizar el primer caso, pero en el material el segundo caso se abusa de modo que los ejemplos son más fáciles de entender.
- En el segundo caso, como ya se indicó, el IDE de Android Studio dará una advertencia en la línea 23 porque recomienda que esa propiedad se declare mediante una constante.
- Las líneas 12 y 19 están creando el ID de los controles para que se pueda acceder a ellos desde Java a través de la Clase R.
- Finalmente observar cómo se define el color azul (RGB), línea 22. Esta definición también podría estar usando una constante declarada en otro archivo XML, como se verá al final de esta sección.

### 1.2.3.Acceder y manipular el control desde Java

- A continuación accederemos al control **TextView** declarado en XML desde Java y realizaremos acciones sobre el control.
- La forma de acceder a cualquier componente gráfico es llamando al método [findViewById \(int id\)](#) en el que pasamos como parámetro el *id* del recurso, que podemos obtener a través de la clase **R**, en la forma: **R.id.resource\_identifier**.

Escribimos estas líneas:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_caja_texto);

    final TextView txtOriginal = findViewById(R.id.txtOriginal_UD02_01_TextView);
}
```

**Nota:** Al poner *final* aplicado a un objeto de una clase impedimos cambiar la referencia de txtOriginal a otro objeto en la clase TextView.

Al hacerlo, si hemos configurado el IDE de Android Studio, importará la clase automáticamente y el *import* aparecerá en la parte superior de la clase, de lo contrario tendremos que hacerlo manualmente:

```
UD02_01_TextView.java ×
1 package olga100.proyectosandroid.pmdm_2020_2021.UI.CajaTexto;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
6 import android.widget.TextView;
7
8 import olga100.proyectosandroid.pmdm_2020_2021.R;
9
10 public class UD02_01_TextView extends AppCompatActivity {
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_caja_texto);
16
17         final TextView txtOriginal = findViewById(R.id.txtOriginal_UD02_01_TextView);
18     }
19 }
```

A medida que vamos escribiendo el tipo si presionamos **CTRL + Barra espaciadora** ya sabemos que nos autocompletará.

- El método `findViewById (int id)` :
  - ✓ **Recibe como parámetro una constante** de la Clase R: en este caso el id asociado a un elemento visual. (CTRL + Barra espaciadora, para ubicar la constante).
  - ✓ **Devuelve un objeto** de tipo **View**.
- Todo control visual es una **subclase** de **View**, por tanto como obtenemos un objeto View.

Antes de la versión 26, sería necesario hacer un *cast* del método de la forma:

```
TextView variable = (TextView) findViewById (R.id.id_text);
```

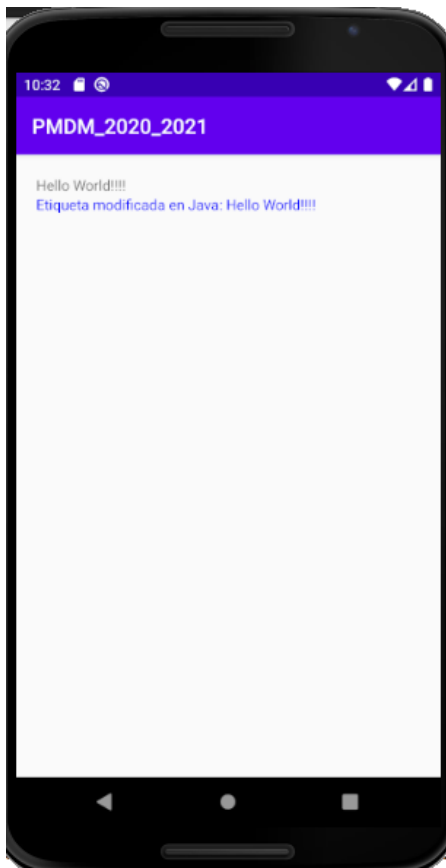
A partir de la versión 26 de API ya no es necesario.

- A continuación se muestra el código Java que va a manejar el TextView:

```
UD02_01_TextView.java
1 package olgal00.proyectosandroid.pmdm_2020_2021.UI.CajaTexto;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
6 import android.widget.TextView;
7
8 import olgal00.proyectosandroid.pmdm_2020_2021.R;
9
10 public class UD02_01_TextView extends AppCompatActivity {
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_caja_texto);
16
17         final TextView txtOriginal = findViewById(R.id.txtOriginal_UD02_01_TextView);
18         final TextView tvJava = (TextView) findViewById(R.id.txtJava_UD02_01_TextView);
19
20         tvJava.setText("Etiqueta modificada en Java: "+txtOriginal.getText());
21     }
22 }
```

- Línea 17: Tenemos un objeto que apunta al TextView original, al primer TextView del Layout.
- Línea 18: Tenemos un objeto que apunta al nuevo TextView, el segundo del Layout.
- Línea 20: Modificamos el texto del segundo TextView. El contenido es una cadena de texto concatenada (+) con el texto que tiene el primer TextView.
  - ✓ Observar la función de los métodos: **setText()** y **getText()** . Estos métodos públicos se pueden ver en el siguiente enlace: <https://developer.android.com/reference/android/widget/TextView.html#pubmethods>
- Cuando se lance la aplicación, se va a ejecutar ese código, con lo cual el texto del segundo TextView no va a ser el que se indico en el Layout en tiempo de diseño sino el que se indica en tiempo de ejecución.
- La imagen muestra la aplicación ejecutándose:



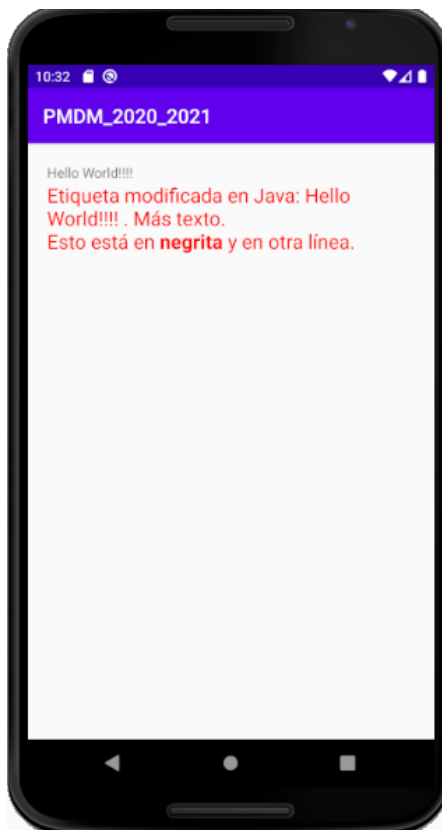


- Observar el contenido de la segunda línea, no es lo que se asignó en el Layout.

### 1.2.4.Manipulación html de una etiqueta de texto

- Las etiquetas de texto no son texto (String) sino que son código similar a html.
- Entonces, en la línea 20 anterior parece haber una contradicción: concatenar una cadena con *algo* html.
- Tendríamos que haber usado el método `toString(): txtOriginal.getText().ToString()`
- Pero en Java no es necesario ponerlo porque ese método se llama automáticamente siempre que el objeto se concatena con otro String (en este caso la cadena de texto).

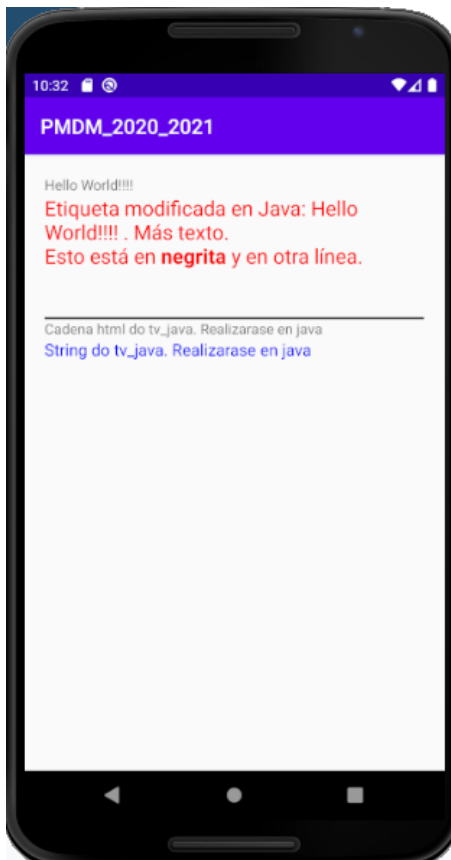
- Para obtener la siguiente imagen no se va a tocar el XML se va a hacer todo en Java.



- En la segunda etiqueta se cambió: color, tamaño de fuente y una palabra en **negrita**.
- Código JAVA...

```
UD02_01_TextView.java
1 package olgal00.proyectosandroid.pmdm_2020_2021.UI.CajaTexto;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.graphics.Color;
6 import android.os.Bundle;
7 import android.text.Html;
8 import android.widget.TextView;
9
10 import olgal00.proyectosandroid.pmdm_2020_2021.R;
11
12 public class UD02_01_TextView extends AppCompatActivity {
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_caja_texto);
18
19         final TextView txtOriginal = findViewById(R.id.txtOriginal_UD02_01_TextView);
20         final TextView tvJava = (TextView) findViewById(R.id.txtJava_UD02_01_TextView);
21
22         tvJava.setText("Etiqueta modificada en Java: "+txtOriginal.getText());
23         tvJava.append(" . Más texto.");
24         tvJava.setTextColor(Color.RED);
25         tvJava.setTextSize(20);
26         tvJava.append(Html.fromHtml("<p><br>Esto está en <b>negrita</b> y en otra línea.</p>", Html.FROM_HTML_MODE_LEGACY));
27
28     }
29 }
```

- Línea 23: agrega contenido a la etiqueta por el final de la etiqueta y muestre el contenido final.
  - Línea 24: cambia de color usando una constante estática.
  - Línea 25: cambiar el tamaño de la fuente
  - Línea 26: Devuelve la cadena en formato HTML en lo que en Android podría llamarse *Texto con estilo mostrable*
- 
- A continuación se va a modificar la aplicación para que podamos recuperar el contenido exacto de un *TextView*, no lo que muestra en pantalla.
  - El método *toString()* también se utilizará para ver su resultado.
- 
- La imagen muestra el diseño del layout.



- Observar la línea de separación y después de esta, dos etiquetas de texto.

- El fichero XML asociado:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:orientation="vertical"
8      android:padding="20sp"
9      tools:context=".UI.CajaTexto.UD02_01_TextView">
10
11      <TextView
12          android:id="@+id/txtOriginal_UD02_01_TextView"
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:layout_marginEnd="15dp"
16          android:text="@string/hello_world" />
17
18      <TextView
19          android:id="@+id/txtJava_UD02_01_TextView"
20          android:layout_width="wrap_content"
21          android:layout_height="wrap_content"
22          android:textColor="#00F"
23          android:text="Da igual lo que escribamos aquí. En este ejemplo,
24              al lanzar la aplicación, se va a cambiar el texto" />
25
26      <View
27          android:layout_width="match_parent"
28          android:layout_height="2sp"
29          android:background="#000" />
30
31      <TextView
32          android:id="@+id/txtTv_html_UD02_01_TextView"
33          android:layout_width="wrap_content"
34          android:layout_height="wrap_content"
35          android:text="Cadena html del tv_java. Se realizará en java" />
36
37      <TextView
38          android:id="@+id/txtTv_string_UD02_01_TextView"
39          android:layout_width="wrap_content"
40          android:layout_height="wrap_content"
41          android:text="String del tv_java. Se realiza en java"
42          android:textColor="#00F"
43          android:textSize="16sp" />
44
45  </LinearLayout>

```

- Observar las líneas marcadas
- La imagen muestra la aplicación en ejecución:



- Observe cómo la primera etiqueta después de la línea tiene el contenido de lo que almacena el EditText rojo.
- La segunda etiqueta después de la línea muestra la etiqueta roja pasada al método **toString** . Observar cómo ya no hay negrita.

- El código que lo hace posible es el siguiente:

```

UD02_01_TextView.java activity_caja_texto.xml
1 package olgal00.proyectosandroid.pmdm_2020_2021.UI.CajaTexto;
2
3
4 import android.graphics.Color;
5 import android.os.Bundle;
6 import android.text.Html;
7 import android.text.Spanned;
8 import android.widget.TextView;
9
10 import androidx.appcompat.app.AppCompatActivity;
11
12 import olgal00.proyectosandroid.pmdm_2020_2021.R;
13
14 public class UD02_01_TextView extends AppCompatActivity {
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_caja_texto);
20
21         final TextView txtOriginal = findViewById(R.id.txtOriginal_UD02_01_TextView);
22         final TextView tvJava = (TextView) findViewById(R.id.txtJava_UD02_01_TextView);
23         final TextView tvHtml = (TextView) findViewById(R.id.txtTv_html_UD02_01_TextView);
24         final TextView tvString = (TextView) findViewById(R.id.txtTv_string_UD02_01_TextView);
25
26         tvJava.setText("Etiqueta modificada en Java: "+txtOriginal.getText());
27         tvJava.append(" . Más texto.");
28         tvJava.setTextColor(Color.RED);
29         tvJava.setTextSize(20);
30         // tvJava.append(Html.fromHtml("<p><br>Esto está en <b>negrita</b> y en otra línea.</p>")); //método OBSOLETO
31         tvJava.append(Html.fromHtml("<p><br>Esto está en <b>negrita</b> y en otra línea.</p>",Html.FROM_HTML_MODE_LEGACY));
32
33         //tvHtml.setText(Html.toHtml((Spanned) tvJava.getText())); // Método OBSOLETO
34         tvHtml.setText(Html.toHtml((Spanned) tvJava.getText(),Html.FROM_HTML_MODE_LEGACY));
35
36         //tvString.setText(""+tvJava.getText());
37         tvString.setText(tvJava.getText().toString());
38     }
39 }
40
41

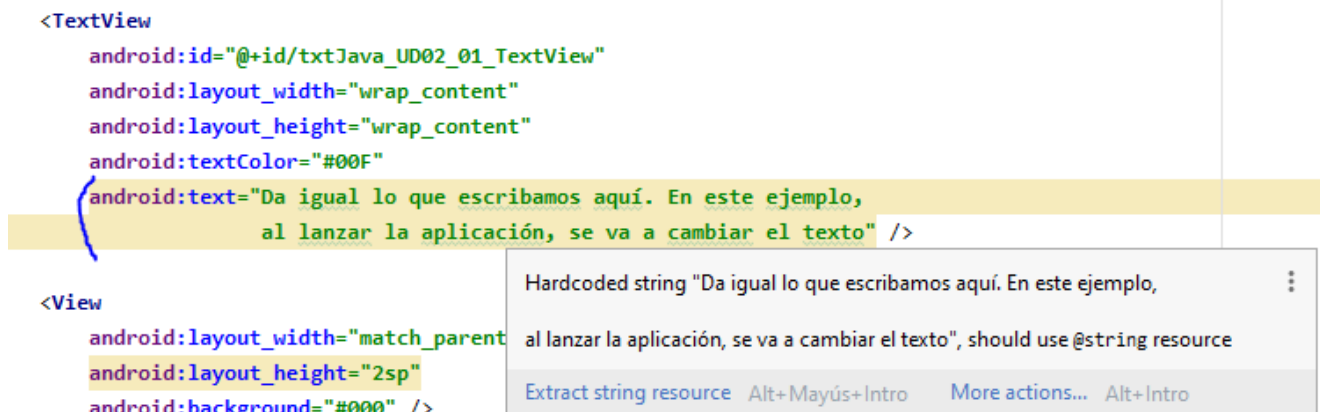
```

- Fijarse que tanto el método *fromHtml* como *toHtml* están obsoletos desde la versión 24 de Android.
- Línea 34: coge el valor de la etiqueta roja y lo pasa a formato HTML.
- Línea 37: pasa la cadena roja a String. Sería equivalente a concatenar una cadena.

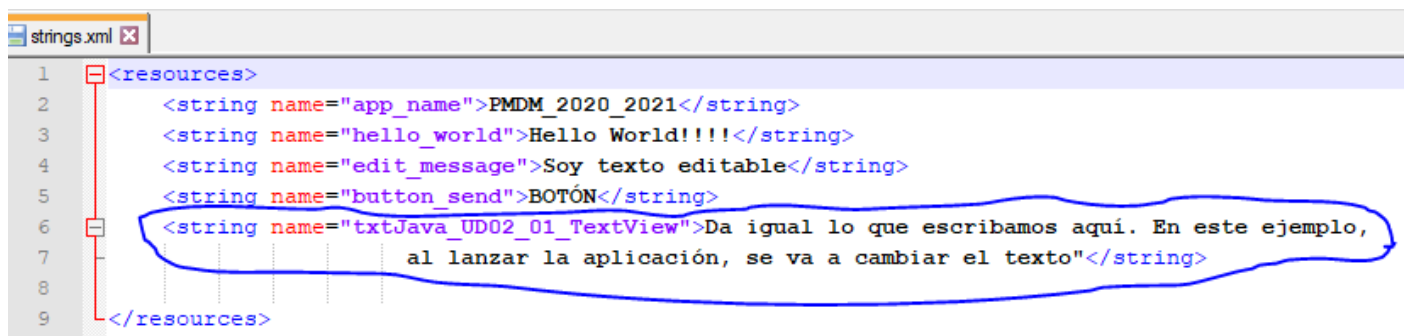
## 1.2.5.Definición de constantes/recursos xml

- Como vimos en la definición XML del diseño, tenemos valores establecidos directamente.
- Sería bueno definir estas propiedades en otros archivos XML, de esta forma se permite la reutilización e internacionalización.

## Recursos string



Advertencia en XML indicando que usemos un recurso de tipo @string



Editamos el fichero **/res/values/strings.xml** o creamos un nuevo en **/res/values**. Añadimos los nuevos recursos de tipo string. Observar las líneas resaltadas. Guardar el fichero.

## Uso recursos string



En el layout hacemos uso de recursos anteriores: (CTRL + Barra espaciadora). Recordar guardar primero el archivo de recursos anterior.

```
<TextView
    android:id="@+id/txtOriginal_UD02_01_TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="15dp"
    android:text="@string/hello_world" />

<TextView
    android:id="@+id/txtJava_UD02_01_TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#00F"
    android:text="@string/txtJava_UD02_01_TextView"/>

<View
    android:layout_width="match_parent"
    android:layout_height="2sp"
    android:background="#000" />

<TextView
    android:id="@+id/txtTv_html_UD02_01_TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txtTv_html_UD02_01_TextView" />

<TextView
    android:id="@+id/txtTv_string_UD02_01_TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txtTv_string_UD02_01_TextView"
    android:textColor="#00F"
    android:textSize="16sp" />
```

Ya esta todo correcto. NOTA: Antes de la versión 2.3 de Android Studio se creaba un archivo '*dimesn.xml*' donde se guardaban constantes que tenían que ver con el tamaño de los márgenes.

➤ A continuación se va a crear un fichero de recurso para los colores.

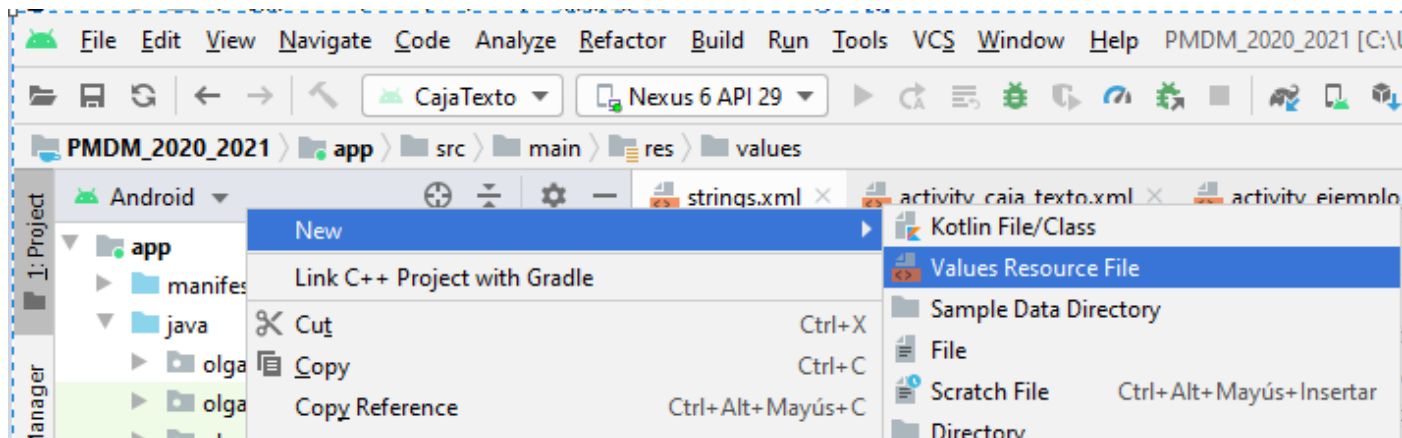
Cada color consta de 3 colores (RED, GREEN, BLUE) que van de 00 a FF en hexadecimal (0 a 255 en decimal), aunque también acepta un solo dígito (de 0 a F).

Normalmente haríamos uso de lo que ya trae por defecto, pero vamos a crear uno nuevo para ver cuál sería el proceso...

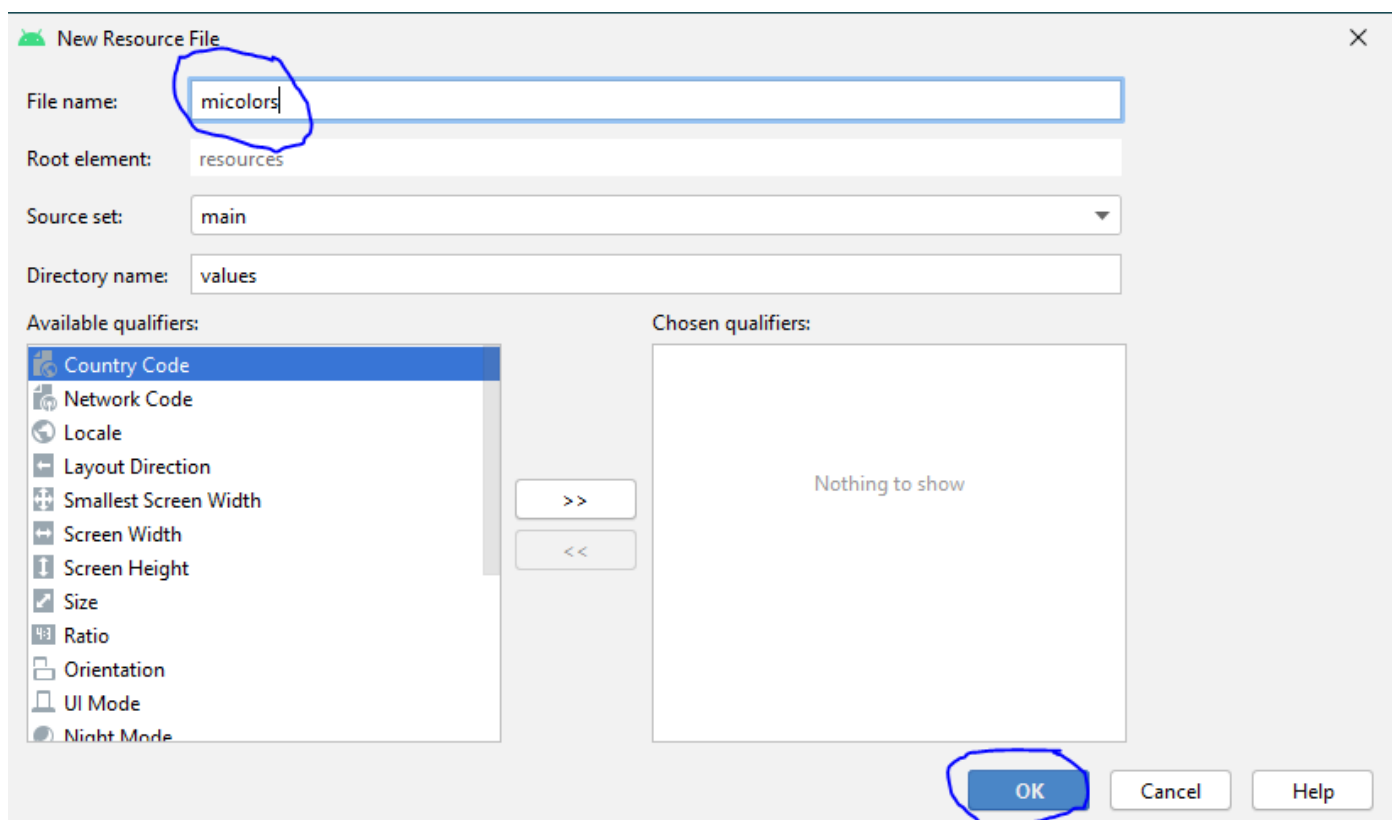


Las constantes de los recursos se pueden crear directamente en el diseñador o bien crearlas previamente y utilizarlas en el diseñador posteriormente.

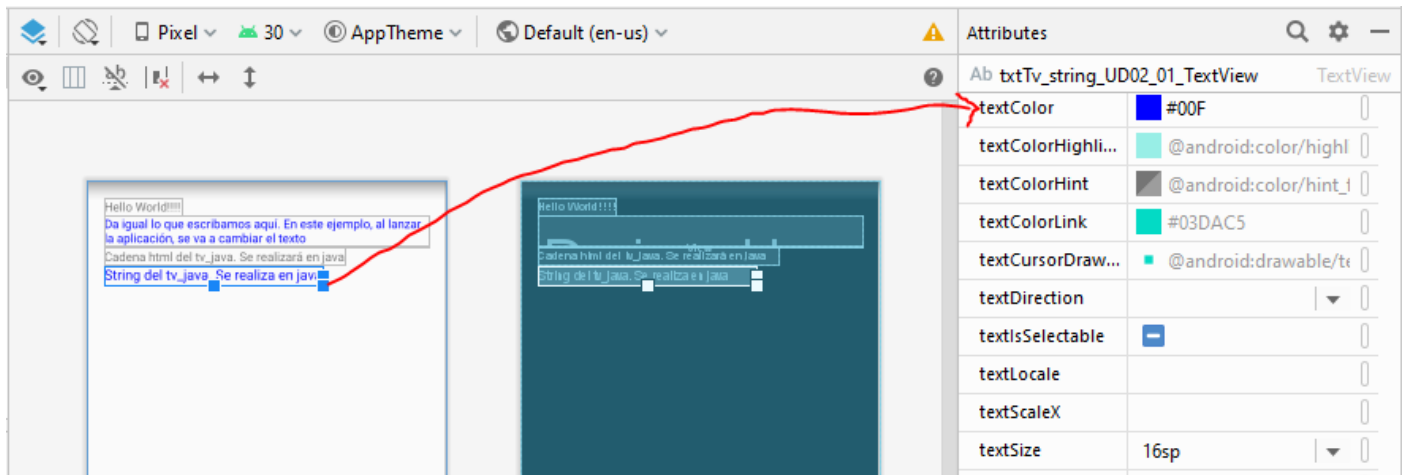
### Crear fichero de recursos



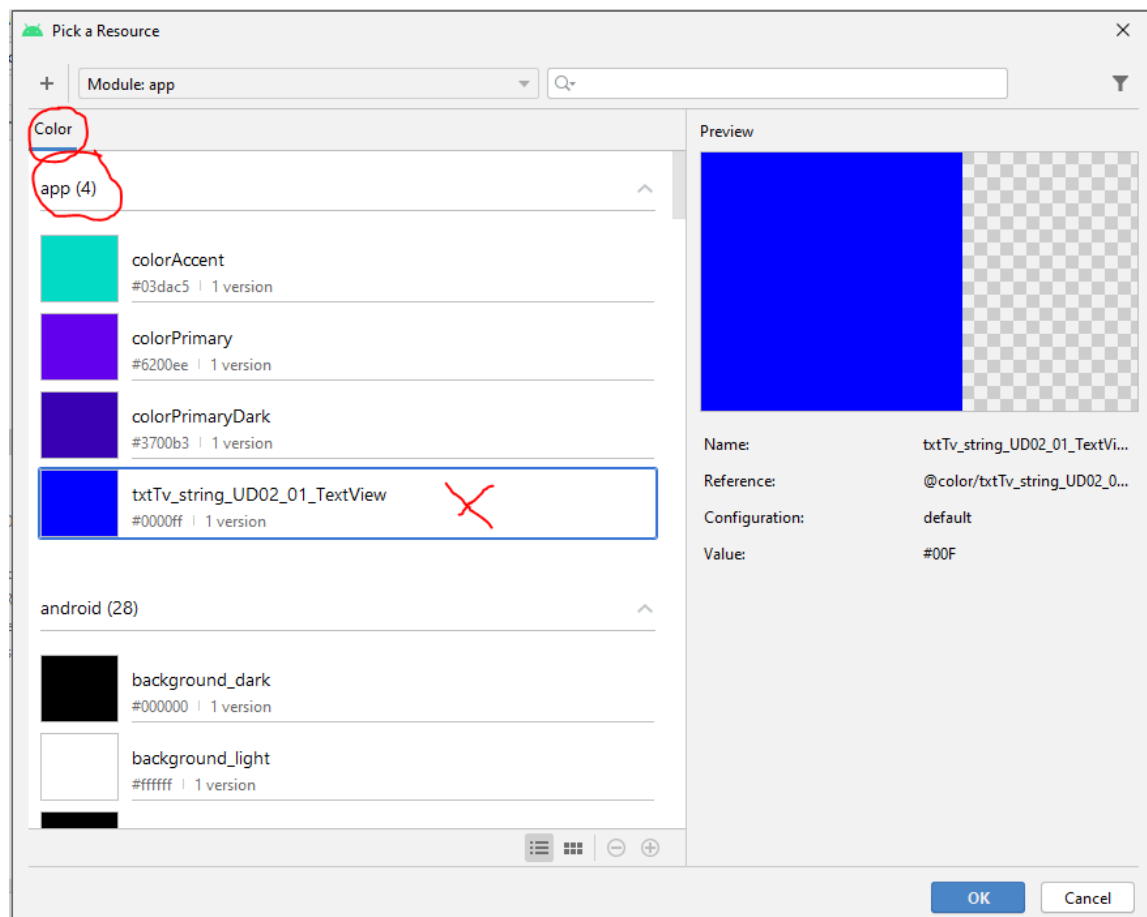
En `/res/values` creamos un nuevo fichero **Values Resource File**



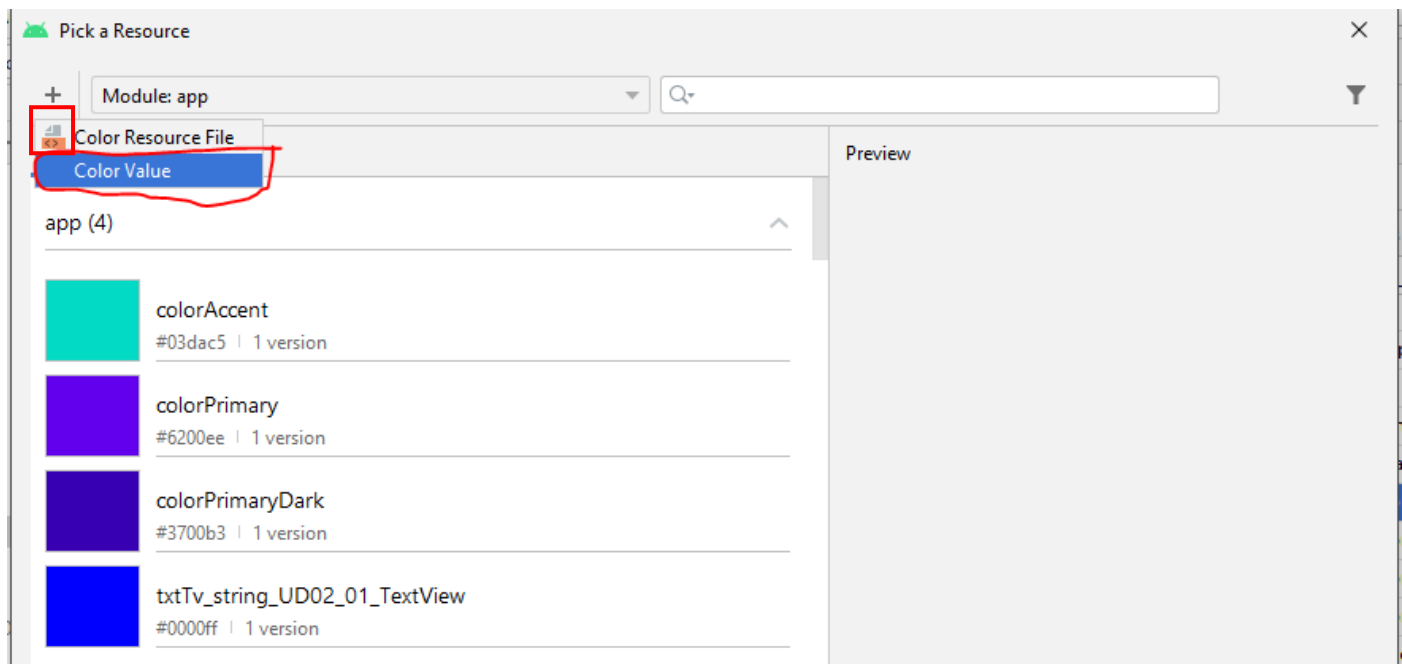
Editamos el archivo y agregamos los valores que queramos. En el ejemplo son colores, por lo que la etiqueta a utilizar es `<color>`. Todos los tipos que podemos usar en `res` (colores, dimensiones, cadena) tienen un atributo de nombre que es el nombre de la constante y un valor entre las etiquetas `<color> valor </color>`.



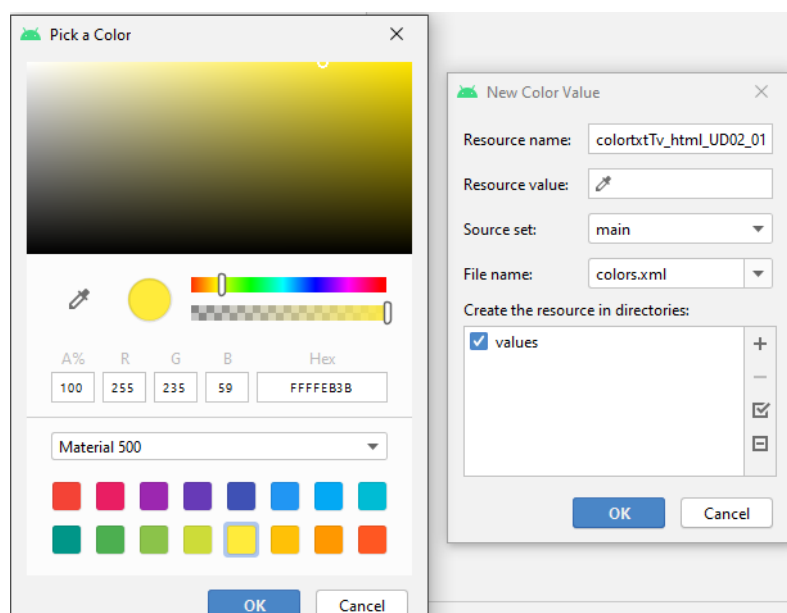
Ahora en el diseñador de layout seleccionamos la view a la que queremos asignar este color y hacemos clic en la barra (elipse) de la derecha.



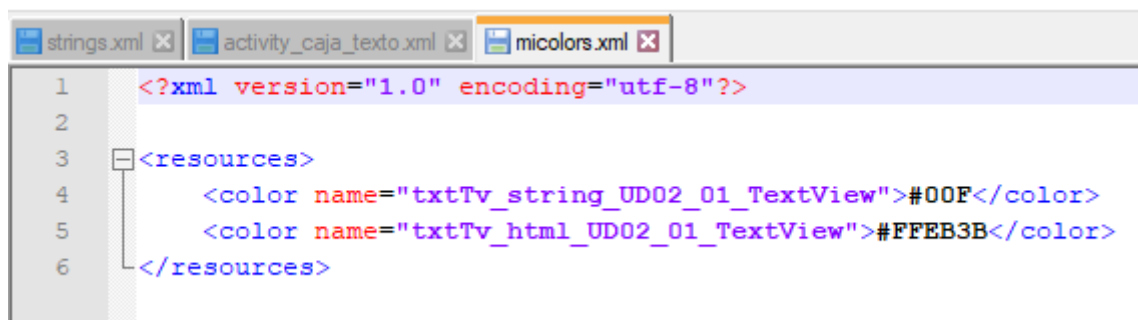
En la sección 'Color' a nivel de 'app', podemos seleccionar el color creado.



Clicamos en el signo + y seleccionamos 'Color Value'



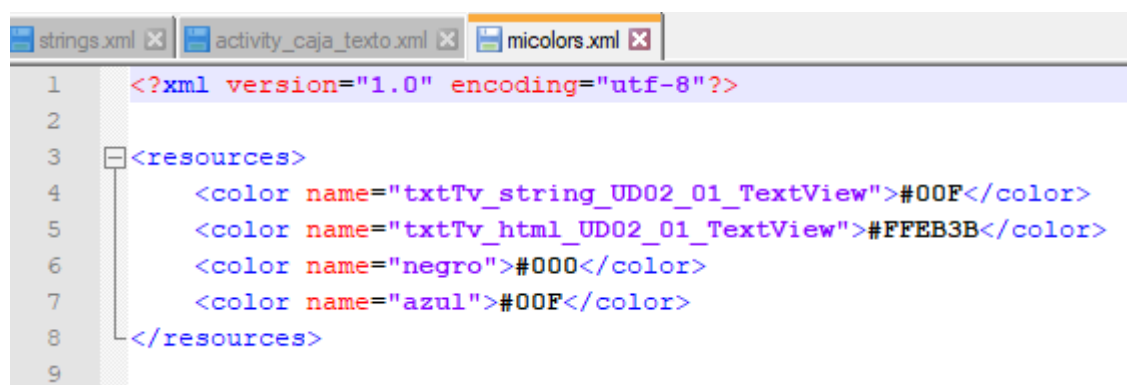
Elegimos el color que queremos usar y lo asociamos a un nombre.



```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <resources>
4     <color name="txtTv_string_UD02_01_TextView">#00F</color>
5     <color name="txtTv_html_UD02_01_TextView">#FFEB3B</color>
6 </resources>
```

El nuevo color se crea automáticamente en el fichero 'micolors.xml'.

➤ Modificamos el archivo de recursos de 'micolors.xml':



```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <resources>
4     <color name="txtTv_string_UD02_01_TextView">#00F</color>
5     <color name="txtTv_html_UD02_01_TextView">#FFEB3B</color>
6     <color name="negro">#000</color>
7     <color name="azul">#00F</color>
8 </resources>
9
```

Uso de recursos en el layout. Observar las líneas marcadas

```
strings.xml x activity_caja_texto.xml x micolors.xml x
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical"
8     android:padding="20sp"
9     tools:context=".UI.CajaTexto.UD02_01_TextView">
10
11     <TextView
12         android:id="@+id/txtOriginal_UD02_01_TextView"
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:layout_marginEnd="15dp"
16         android:text="@string/hello_world" />
17
18     <TextView
19         android:id="@+id/txtJava_UD02_01_TextView"
20         android:layout_width="wrap_content"
21         android:layout_height="wrap_content"
22         android:textColor="@color/azul"
23         android:text="@string/txtJava_UD02_01_TextView"/>
24
25
26     <View
27         android:layout_width="match_parent"
28         android:layout_height="2sp"
29         android:background="@color/negro" />
30
31     <TextView
32         android:id="@+id/txtTv_html_UD02_01_TextView"
33         android:layout_width="wrap_content"
34         android:layout_height="wrap_content"
35         android:text="@string/txtTv_html_UD02_01_TextView"
36         android:textColor="@color/txtTv_html_UD02_01_TextView" />
37
38     <TextView
39         android:id="@+id/txtTv_string_UD02_01_TextView"
40         android:layout_width="wrap_content"
41         android:layout_height="wrap_content"
42         android:text="@string/txtTv_string_UD02_01_TextView"
43         android:textColor="@color/azul"
44         android:textSize="16sp" />
45
46 </LinearLayout>
47
```

**NOTA IMPORTANTE:** recordar que cualquier recurso a crear solo aceptará letras minúsculas y guión bajo. Se prohíben otros caracteres.

Anda!!! Hemos dejado en la línea 24 y dos valores candidatos a definir en constantes de tipo *dimen* . Seguro que el estudiante podrá hacerlo.

La etiqueta de las dimensiones es: `<dimen> </dimen>`. Recordar utilizar el atributo '*name*'.

Los recursos de tamaño son buenos tenerlos para aplicar el mismo tamaño a todas las views de nuestra aplicación y de esa forma daremos la impresión de tener un mismo diseño.

Importante: El tamaño que se guarde en `<dimen>` ten que especificar a unidad, *sp* ou *dp* da forma: `<dimen name='nome'>10sp</dimen>`

## 1.2.6.Clickable / No Clickable

- Todas las view's tienen la propiedad '*Clickable*' en la documentación indica que sirve para que la view responda a los eventos de Click sobre el mismo.

El 'problema' que tenemos es que al gestionar el evento Click desde el código, al poner el comando *setOnClickListener* (*new OnClickListener*) volvemos a hacer clickable el control.

Si no queremos que esto no suceda, tendremos que deshabilitar el clickable desde el código de la siguiente manera:

```
1      tvJava.setOnClickListener(new View.OnClickListener() {
2          @Override
3          public void onClick(View v) {
4              TextView t = (TextView)v;
5              t.setText("OLA");
6          }
7      });
8      tvJava.setClickable(false);
```

Como vemos, la orden *setClickable* tiene que estar después del registro de eventos.

Lo mismo sucede con cualquier View.

- Otro aspecto a tener en cuenta es cuando tenemos direcciones web, correos electrónicos, mapas, como parte del texto que se muestra.

En ese caso entrarían en juego las propiedades **autolink** y **linkclickables**.

- Por ejemplo:

```
1 tvJava.setText("https://www.google.es");
```

- Por último, si queremos mostrar enlaces html en el TextView dentro del código de la Activity de la forma:

```
1 tvJava.append(Html.fromHtml("<br /><a href=\"http://www.google.com\">This is a link</a>"));
2 tvJava.setMovementMethod(LinkMovementMethod.getInstance());
```

- Tenemos que añadir la línea 2 para que lo haga. Más información en este [enlace](#).

## 1.3. Accesibilidad

- Podemos activar el modo *TalkBack* siguiendo las instrucciones de este [enlace](#).

Este modo "leerá" la descripción de cada elemento.

- Como regla general podemos dar las siguientes pautas:

Un <TextView> deberá tener el atributo '**android:contentDescription**' como se muestra en el siguiente ejemplo:

```
<TextView
    android:id="@+id/textView7"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Dime tu nombre:"
    android:textSize="18sp"
    android:contentDescription="Nombre a enviar"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="@+id/etNome_UD03_01_Intents" />
```

- En caso de que TextView esté asociado con EditText, podemos reemplazar este atributo con el atributo: **android:labelFor** que indica cuál es el EditText asociado con TextView.

```

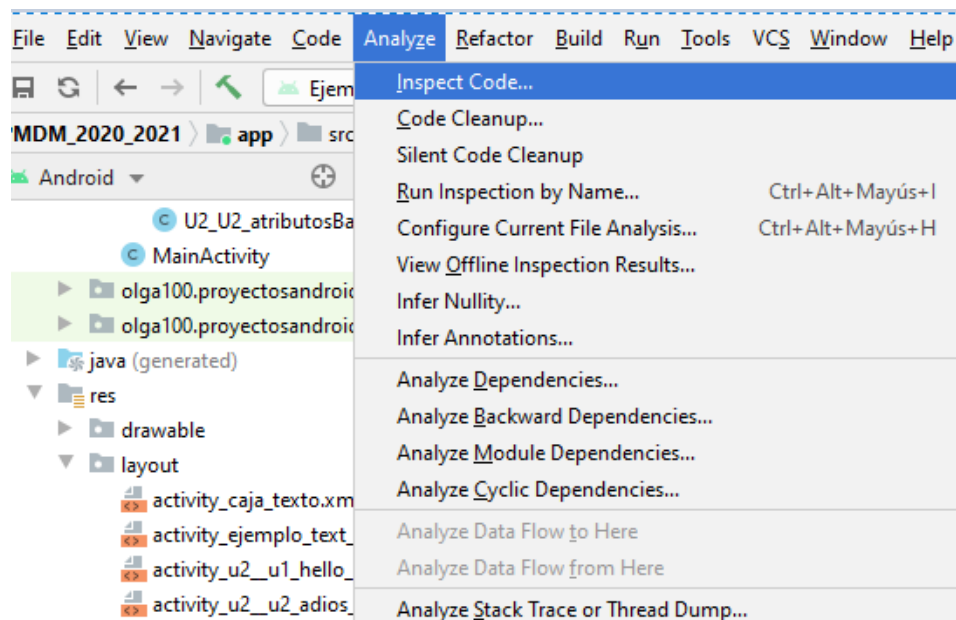
<TextView
    android:id="@+id/textView7"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Dime tu nombre:"
    android:textSize="18sp"
    android:labelFor="@id/etNome_UD03_01_Intents"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="@+id/etNome_UD03_01_Intents" />

<EditText
    android:id="@+id/etNome_UD03_01_Intents"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:ems="10"
    android:inputType="textPersonName"
    app:layout_constraintStart_toEndOf="@+id/textView7"
    app:layout_constraintTop_toTopOf="parent" />

```

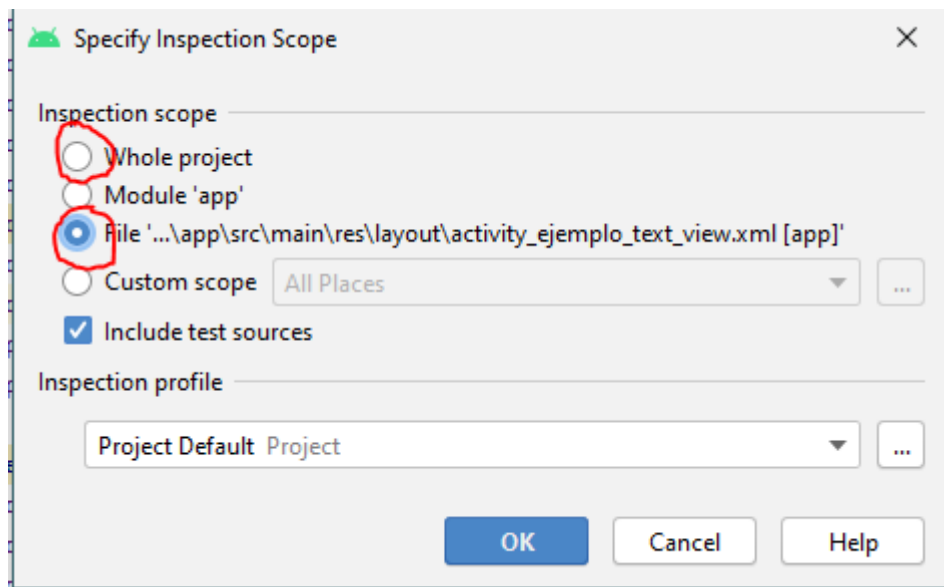
- Para comprobar que no tenemos problemas de accesibilidad:

### Comprobando la accesibilidad

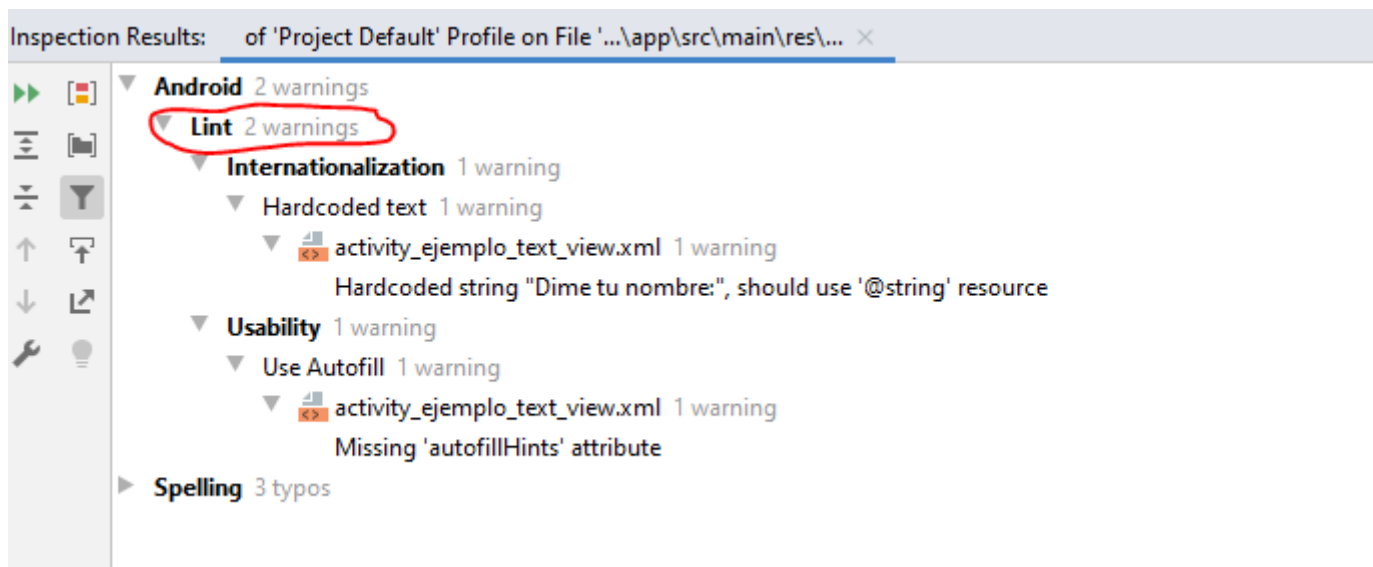


- Vamos al menú de Android Studio y escogemos la opción **Analyze => Inspect Code**.





Indicamos si queremos analizar el proyecto completo o la activity actual



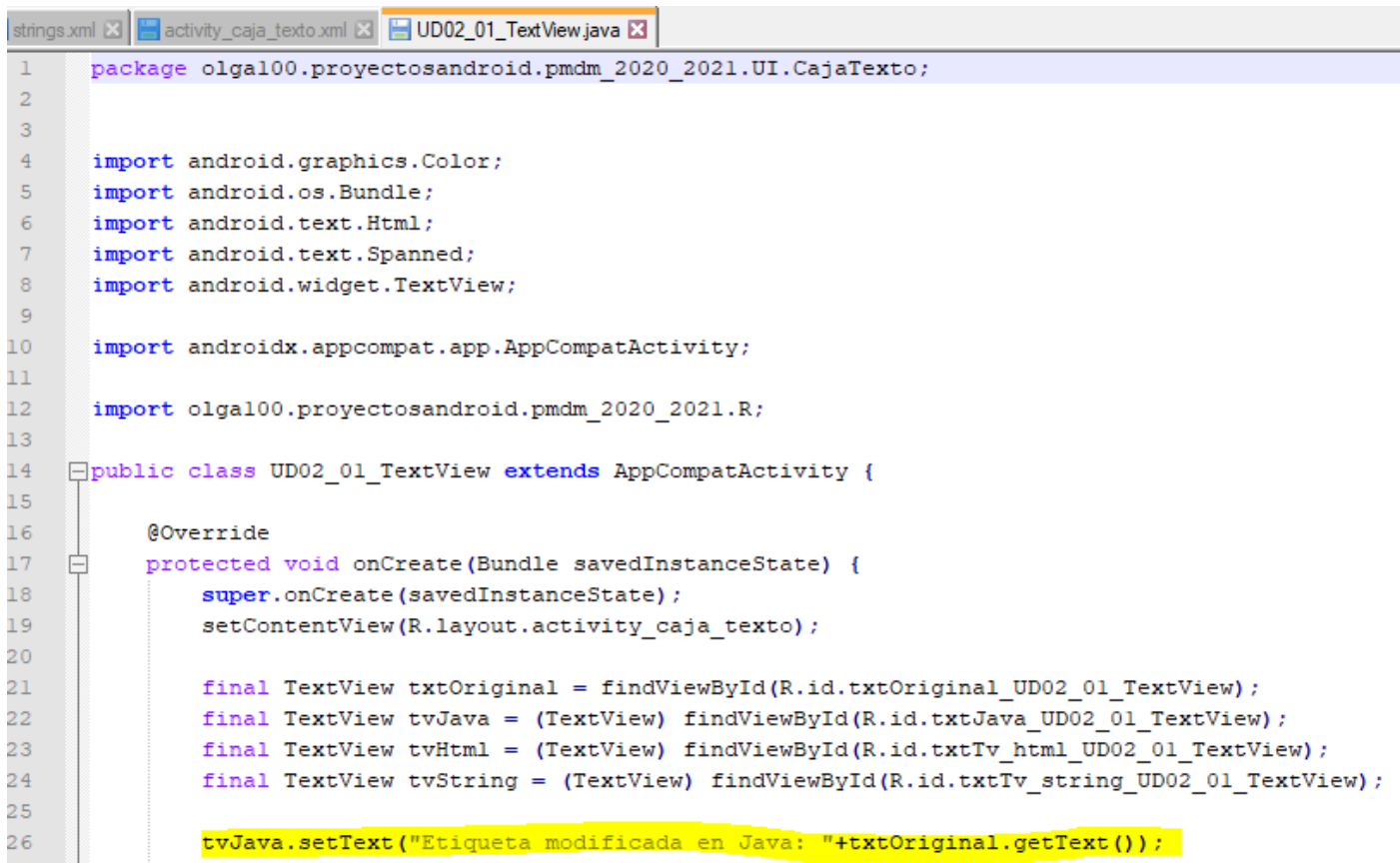
## 1.4. Empleando recursos definidos Res

- Veremos en el tema de internacionalización cómo podemos tener cadenas en varios idiomas y que Android elige la correcta en función del idioma elegido a nivel de sistema operativo en los ajustes.

Pero hay otras cadenas que podemos usar directamente en el código de la aplicación y no a través de **/res/layout** de la activity.

Esas cadenas también necesitarán que estén traducidas en diferentes idiomas (si nuestra aplicación tiene esa funcionalidad).

Para hacer esto, no podemos 'teclear' directamente la cadena de texto en el código como hicimos en ejercicios anteriores, de la siguiente manera:



```

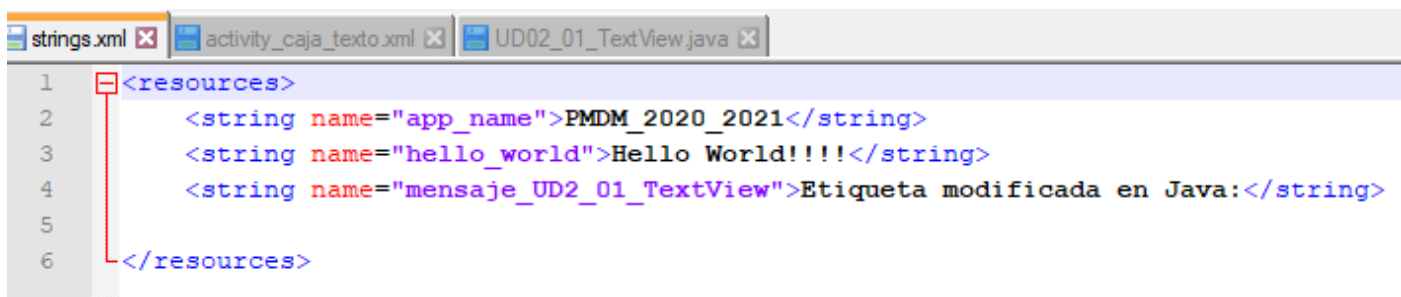
1 package olgal00.proyectosandroid.pmdm_2020_2021.UI.CajaTexto;
2
3
4 import android.graphics.Color;
5 import android.os.Bundle;
6 import android.text.Html;
7 import android.text.Spanned;
8 import android.widget.TextView;
9
10 import androidx.appcompat.app.AppCompatActivity;
11
12 import olgal00.proyectosandroid.pmdm_2020_2021.R;
13
14 public class UD02_01_TextView extends AppCompatActivity {
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_caja_texto);
20
21         final TextView txtOriginal = findViewById(R.id.txtOriginal_UD02_01_TextView);
22         final TextView tvJava = (TextView) findViewById(R.id.txtJava_UD02_01_TextView);
23         final TextView tvHtml = (TextView) findViewById(R.id.txtTv_html_UD02_01_TextView);
24         final TextView tvString = (TextView) findViewById(R.id.txtTv_string_UD02_01_TextView);
25
26         tvJava.setText("Etiqueta modificada en Java: "+txtOriginal.getText());

```

Como vemos, tenemos el texto 'Etiqueta modificada en Java:' escrito directamente en el código.

- Lo que tenemos que hacer es crear una constante en un archivo de recursos en *'/res/values/'* o usar una ya creada, por ejemplo *strings.xml*.

Dentro de este archivo (u otro creado) definimos la constante con el texto asociado del formulario:



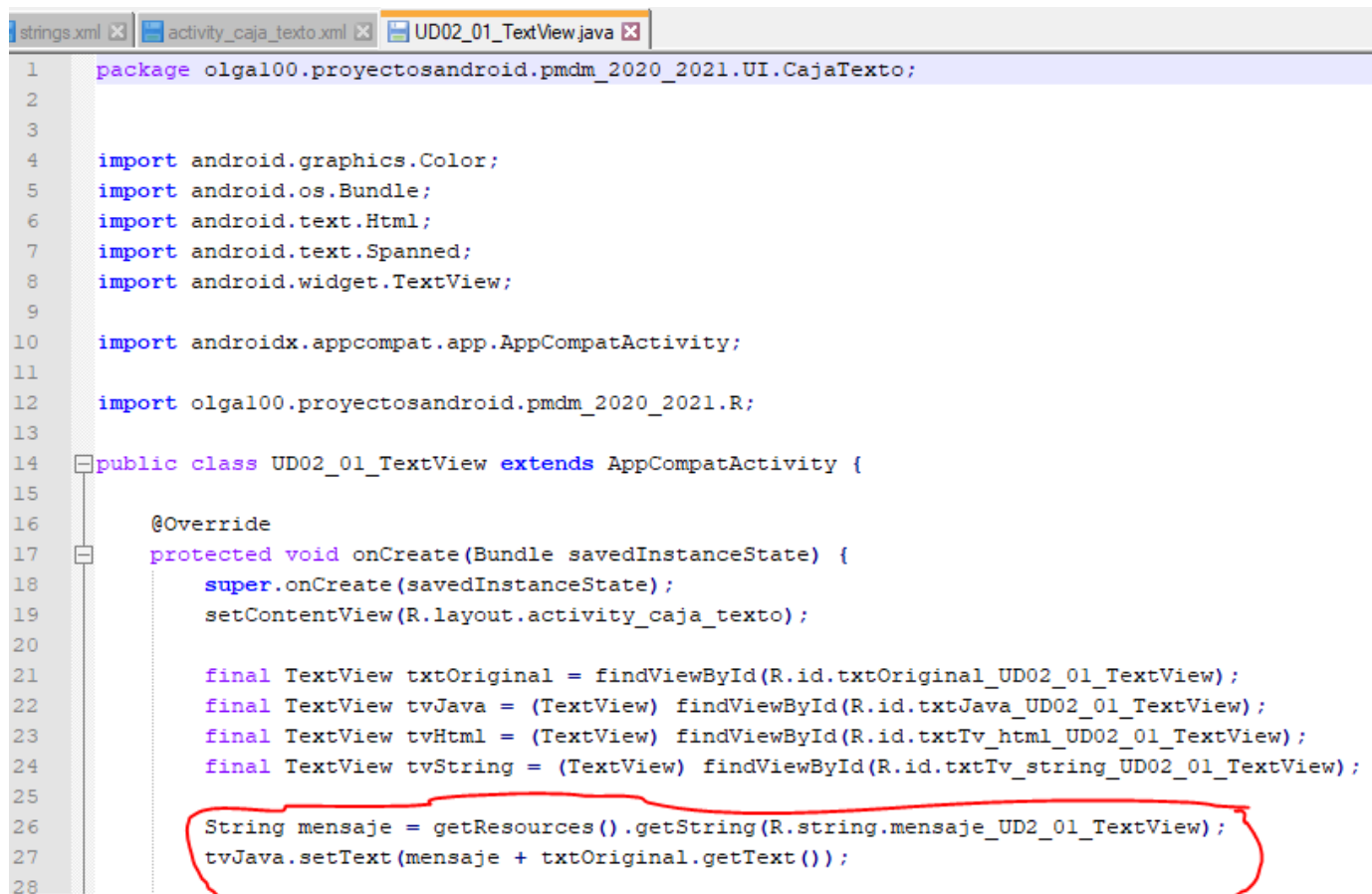
```

1 <resources>
2     <string name="app_name">PMDM_2020_2021</string>
3     <string name="hello_world">Hello World!!!!</string>
4     <string name="mensaje_UD2_01_TextView">Etiqueta modificada en Java:</string>
5
6 </resources>

```

Como vemos, el nombre de la constante es: *mensaje\_UD2\_01\_TextView*

- Ahora, para usar esta constante en código Java, tenemos que usar el [método getResources\(\)](#) de la siguiente manera:



```

1  package olgal00.proyectosandroid.pmdm_2020_2021.UI.CajaTexto;
2
3
4  import android.graphics.Color;
5  import android.os.Bundle;
6  import android.text.Html;
7  import android.text.Spanned;
8  import android.widget.TextView;
9
10 import androidx.appcompat.app.AppCompatActivity;
11
12 import olgal00.proyectosandroid.pmdm_2020_2021.R;
13
14 public class UD02_01_TextView extends AppCompatActivity {
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_caja_texto);
20
21         final TextView txtOriginal = findViewById(R.id.txtOriginal_UD02_01_TextView);
22         final TextView tvJava = (TextView) findViewById(R.id.txtJava_UD02_01_TextView);
23         final TextView tvHtml = (TextView) findViewById(R.id.txtTv_html_UD02_01_TextView);
24         final TextView tvString = (TextView) findViewById(R.id.txtTv_string_UD02_01_TextView);
25
26         String mensaje = getResources().getString(R.string.mensaje_UD2_01_TextView);
27         tvJava.setText(mensaje + txtOriginal.getText());
28     }

```

- **Nota:** El método `setText` está sobrecargado y permite que un recurso `String` se use como datos directamente desde el formulario: `R.string.resource_name`, pero solo podemos hacer esto si no agregamos una nueva cadena del formulario: `tvJava.setText(R.string.resource_name + "nueva cadena")`. En este caso, debemos usar la opción anterior para obtener primero la cadena definida en el recurso o usar el método `append()`. Recuerde que `R.resource_type.resource_name` es un número definido en la clase `R`. Solo aquellos métodos que aceptan un parámetro de tipo `'int resId'` traducirán ese número al recurso definido en `/res/values/`.

## 1.5. Atributos que debes conocer

- Tenemos una lista de atributos en <https://developer.android.com/reference/android/widget/TextView>
- Dentro de este control tenéis que conocer y saber emplear los siguientes atributos:
- ✓ `id`
  - ✓ `layout_width`, `layout_height`
  - ✓ `margin`
  - ✓ `padding`
  - ✓ `gravity`

- ✓ laberFor
- ✓ enabled
- ✓ lines
- ✓ maxlength
- ✓ text
- ✓ textXXXX (Size, Color, Style, Alignment)
- ✓ fontFamily
- ✓ typeFace
- ✓ visibility

## 1.6. Métodos mínimos que debes conocer en el manejo de TextViews

- Referenciar a un TextView con el método *findViewById*.
- Recupera el contenido del texto.
- Cambia el contenido del texto, pudiendo utilizar recursos guardados en /res /values.
- Agrega texto nuevo a uno existente.
- Modificar propiedades básicas, como color, tamaño, visibilidad, ... (y métodos getZZZZZ para obtener sus valores)
- Administrar el evento Click en cualquier TextView y saber cómo hacerlo usando interfaces anónimas o implementando la interface en la activity (veremos posteriores temas).

```

TextView tv = findViewById(R.id.txvTextoTextView); // Referenciamos a un TextView que este en la Activity.

String cadenatv = tv.getText().toString();          // Guarda la cadena del TextView''

tv.setText("Cambiamos el texto directamente");      // Cambiamos el contenido por una cadena concreta
tv.setText(R.string.app_name);                     // Cambiamos el contenido por una cadena guardada en un archivo de recursos en /res/values/.
                                                    //De esta forma el texto puede traducirse.
String textoRes = getResources().getString(R.string.app_name);
tv.setText(textoRes + " texto que viene de la BD"); // Si queremos concatenar texto de una base de datos con un texto que pueda ser traducido

tv.append(" añadimos nuevo texto");                // append para añadir

tv.setTextSize(20);                               // Ajustar tamaño

tv.setTextColor(Color.BLUE);                      // Cambia el color

tv.setVisibility(View.VISIBLE);                   // Las constantes aparecen en Android Studio al teclear. GONE hace que no ocupe espacio en el Layout.

tv.setOnClickListener(new View.OnClickListener() { // Gestionamos el evento de Click con una interface anónima
    @Override
    public void onClick(View view) {
        Toast.makeText(getApplicationContext(), "TextView Pulsado", Toast.LENGTH_LONG).show();
    }
});

```