

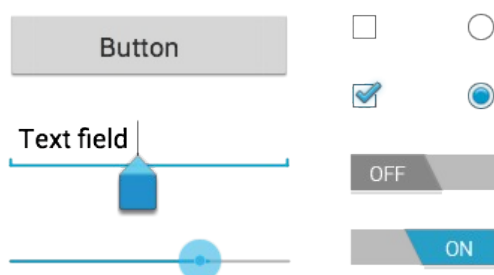
INDICE

1.1. CONTROLES	1
1.2. TEXTVIEW. DEFINICIÓN DE RECURSOS XML	2
1.2.1. INTRODUCCIÓN	2
1.2.2. CASOS PRÁCTICOS	3
1.2.3. ACCEDER Y MANIPULAR EL CONTROL DESDE JAVA	5
1.2.4. MANIPULACIÓN HTML DE UNA ETIQUETA DE TEXTO	7
1.2.5. DEFINICIÓN DE CONSTANTES/RECURSOS XML	10
1.2.6. CLICKABLE / NO CLICKABLE	16
1.3. ACCESIBILIDAD	16
1.4. EMPLEANDO RECURSOS DEFINIDOS RES	18
1.5. ATRIBUTOS QUE DEBES CONOCER	19
1.6. MÉTODOS QUE DEBES CONOCER EN EL MANEJO DE TEXTVIEWS	20

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**.
 - Se puede agregar un control a un layout a través del componente XML o mediante Java en tiempo de ejecución.
 - En las siguientes secciones iremos usando 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 estudiante estudiar aquellos controles en los que está interesado en el paquete anterior y no están contemplados en el tema.

A continuación, 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>

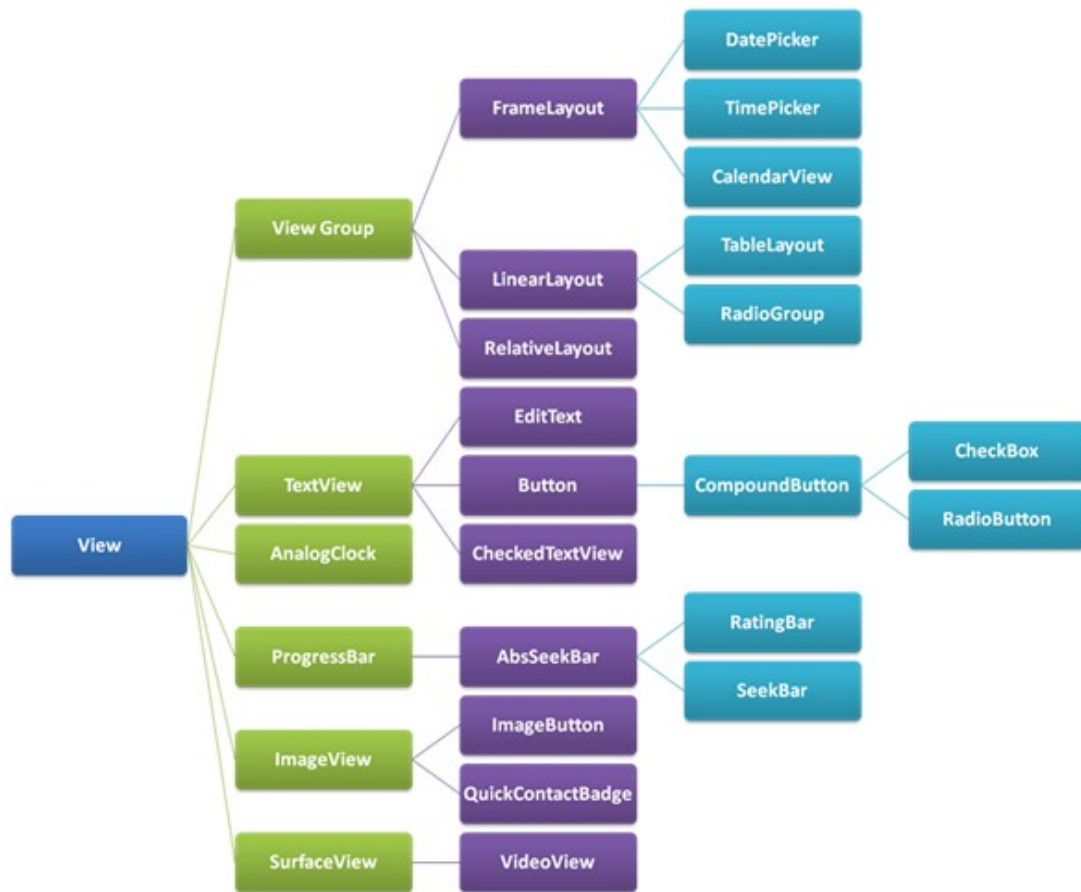
1.2. TextView. Definición de recursos XML

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 memorizarlo) 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

Imagen obtenida de: <http://www.itcsolutions.eu/2011/08/27/android-tutorial-4-procedural-vs-declarative-design-of-user-interfaces>

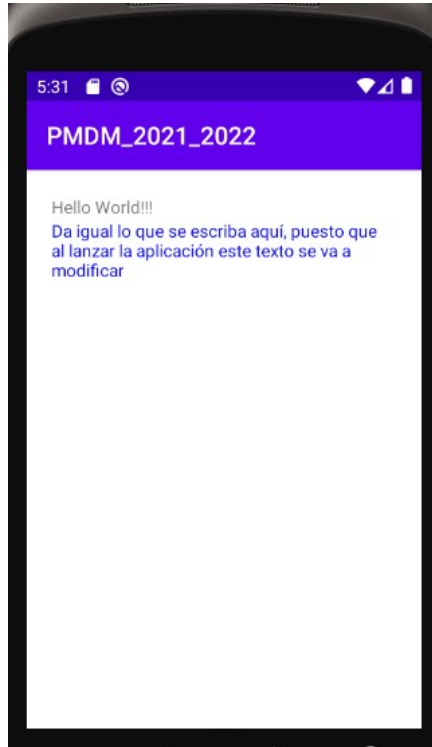
- ❖ Control **TextView**: <https://developer.android.com/reference/android/widget/TextView.html>
 - Observar en el enlace anterior el valor del atributo **editable**.
- ❖ Recomendaciones de tipografía: <https://material.io/design/typography/the-type-system.html#type-scale>
 - Observar en el enlace de 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 en el tema 2.3 (la app llamada PMDM_2021_2022). 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.

- Chequear si es necesario modificar el archivo **AndroidManifest.xml** para agregar una 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. Leed el contenido de la etiqueta azul.

El código XML del layout asociado con esa imagen es el siguiente:

```

activity_ud0201_text_view.xml x strings.xml x UD02_01_TextView.java x
Code Split D
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/tv_original"
13     android:layout_width="wrap_content"
14     android:layout_height="wrap_content"
15     android:text="@string/hello_world" />
16
17   <TextView
18     android:id="@+id/tv_java"
19     android:layout_width="wrap_content"
20     android:layout_height="wrap_content"
21     android:textColor="#00F"
22     android:text="Da igual lo que se escriba aquí, puesto que al lanzar la aplicación este texto
23   </LinearLayout>

```

- En las líneas 12 y 18 se asocia un ID a cada control TextView, que luego se usará en Java a través de la Clase R.
- Observad las diferencias entre las líneas 15 y 22. La primera mostrará el texto que contiene la constante definida en XML (en otro archivo XML), la segunda muestra el texto directamente.
- La definición de la constante **@string/hello_world** está en el archivo de recursos: **/res/values/strings.xml**, como se muestra después de la imagen.



- Como ya se ha mencionado, es recomendable utilizar el primer caso, definir las constantes en el fichero de recursos, pero en el material del curso se abusará del otro modo para que los ejemplos sean más fáciles de entender. En el segundo caso, como ya se indicó, el IDE de Android Studio dará una advertencia en la línea 22 porque recomienda que esa propiedad se declare mediante una constante.
- Finalmente observad cómo se define el color azul (RGB), en la línea 21. Esta definición también podría estar usando una constante declarada en otro archivo XML, **/res/values/colors.xml**, como se verá al final de este tema.

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 una constante de la clase R: el *id* del elemento visual, que podemos obtener a través de la clase **R**, en la forma: **R.id.resource_identifier**. El método `findViewById()` devuelve un objeto de tipo View.

Escribimos estas líneas:

```
package inesvalino.iessanclemente.pmdm_2021_2022.UI.CajaTexto;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

import inesvalino.iessanclemente.pmdm_2021_2022.R;

public class UD02_01_TextView extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ud0201_text_view);
        final TextView tvOriginal = (TextView) findViewById(R.id.tv_original);
    }
}
```

Nota: Al poner **final** aplicado a un objeto de una clase impedimos cambiar la referencia de `tvOriginal` 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:

```
import android.widget.TextView;
```

- ❖ Todo control visual es una **subclase** de **View**, por lo tanto obtenemos un objeto View. Antes de la versión 26, era 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 (por eso os aparece en gris, al ser innecesario).

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



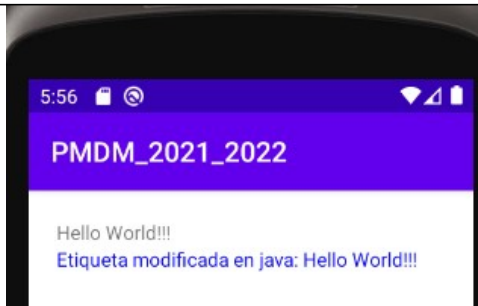
```

1 package inesvalino.iessanclemente.pmdm_2021_2022.UI.CajaTexto;
2
3 import ...
4
5
6
7
8
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_ud0201_text_view);
16
17         final TextView tvOriginal = findViewById(R.id.tv_original);
18         final TextView tvJava = findViewById(R.id.tv_java);
19
20         tvJava.setText("Etiqueta modificada en java: "+tvOriginal.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.
 - Observad 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 indicó en el Layout en tiempo de diseño sino el que se indica en tiempo de ejecución.

La imagen muestra la aplicación ejecutándose:



Observad como 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 este 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**.

El correspondiente código JAVA es:

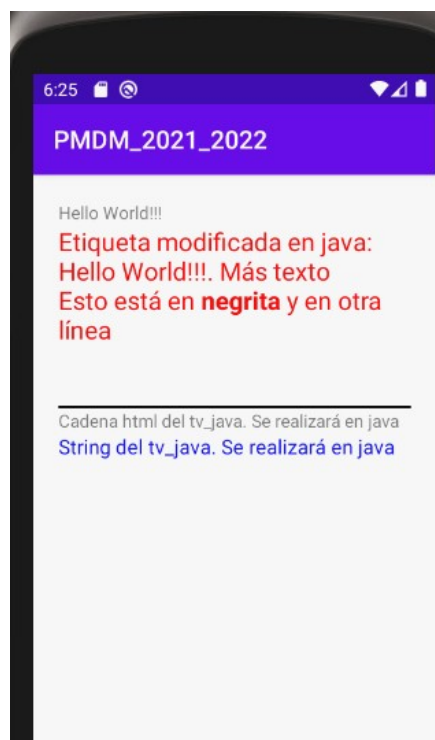
```

11 UD02_01_TextView.java
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_ud0201_text_view);
18
19         final TextView tvOriginal = findViewById(R.id.tv_original);
20         final TextView tvJava = findViewById(R.id.tv_java);
21
22         tvJava.setText("Etiqueta modificada en java: "+tvOriginal.getText());
23         tvJava.append(". Más texto");
24         tvJava.setTextColor(Color.RED);
25         tvJava.setTextSize(20);
26         tvJava.append(Html.fromHtml( source: "<p><br>Esto está en <b>negrita</b> y en otra línea </p>",Html.FROM_HTML_MODE_LEGACY));
27     }
28 }

```

- Línea 23: agrega contenido a la etiqueta por el final de la misma y muestra el contenido final.
 - Línea 24: cambia de color usando una constante estática (Color.*RED*).
 - Línea 25: cambia el tamaño de la fuente
 - Línea 26: Devuelve la cadena en formato HTML en lo que en Android se podría llamar *Texto con estilo mostrable*
- ❖ A continuación se va a modificar la aplicación para que podamos recuperar el contenido exacto de un *TextView*, no el que muestra en pantalla. También se usará el método **toString()** para ver su resultado.

La siguiente imagen muestra el diseño del layout:



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

El fichero XML asociado es:

```

<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout 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"
android:orientation="vertical"
android:padding="20sp"
tools:context=".UI.CajaTexto.UD02_01_TextView">

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

<TextView
    android:id="@+id/tv_java"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#00F"
    android:text="Da igual lo que se escriba aquí, puesto que al lanzar la aplicación este texto se
va a modificar" />

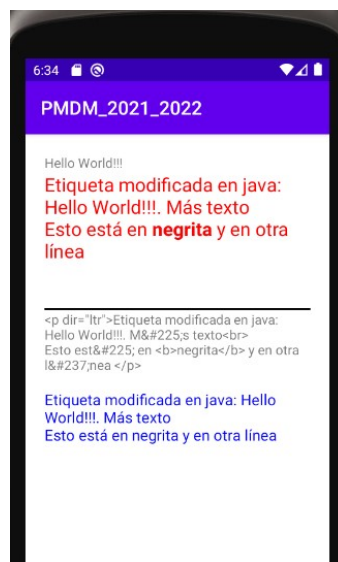
<View
    android:layout_width="match_parent"
    android:layout_height="2sp"
    android:background="#000" />

<TextView
    android:id="@+id/tv_html"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Cadena html del tv_java. Se realizará en java" />;

<TextView
    android:id="@+id/tv_string"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="String del tv_java. Se realizará en java"
    android:textColor="#00F"
    android:textSize="16sp" />
</LinearLayout>

```

Observar las líneas marcadas en amarillo. Tal y como podéis intuir, se quiere que la primera etiqueta después de la línea tenga el contenido de lo que almacena el EditText rojo, y que la segunda etiqueta tras la línea muestre la etiqueta roja pasada por el método **toString**. La aplicación en ejecución debería ser como la siguiente imagen:



Observad como en esta segunda etiqueta tras la línea ya no hay texto en negrita.

❖ El código que lo hace posible es el siguiente:

```

package inesvalino.iessanclemente.pmdm_2021_2022.UI.CajaTexto;

import androidx.appcompat.app.AppCompatActivity;

import android.graphics.Color;
import android.os.Bundle;
import android.text.Html;
import android.text.Spanned;
import android.widget.TextView;

```

```
import inesvalino.iessanclemente.pmdm_2021_2022.R;

public class UD02_01_TextView extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ud0201_text_view);

        final TextView tvOriginal = findViewById(R.id.tv_original);
        final TextView tvJava = findViewById(R.id.tv_java);
        final TextView tvHtml = findViewById(R.id.tv_html);
        final TextView tvString = findViewById(R.id.tv_string);

        tvJava.setText("Etiqueta modificada en java: "+tvOriginal.getText());
        tvJava.append(". Más texto");
        tvJava.setTextColor(Color.RED);
        tvJava.setTextSize(20);
        tvJava.append(Html.fromHtml("<p><br>Esto está en <b>negrita</b> y en otra línea </p>", Html.FROM_HTML_MODE_LEGACY));

        tvHtml.setText(Html.toHtml((Spanned) tvJava.getText(), Html.FROM_HTML_MODE_LEGACY));
        tvString.setText(tvJava.getText().toString());
    }
}
```

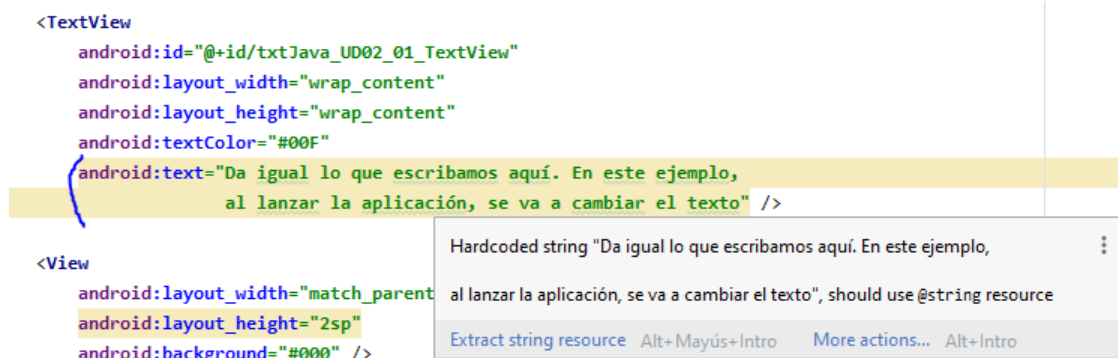
- Línea 31: coge el valor de la etiqueta roja y lo pasa a formato HTML.
- Línea 33: pasa la cadena roja a String. Sería equivalente a concatenar una cadena.

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



```
<TextView
    android:id="@+id/txtJava_UD02_01_TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#00F"
    android:text="Da igual lo que escribamos aquí. En este ejemplo,
    al lanzar la aplicación, se va a cambiar el texto" />

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

Hardcoded string "Da igual lo que escribamos aquí. En este ejemplo, al lanzar la aplicación, se va a cambiar el texto", should use @string resource

Extract string resource Alt+Mayús+Intro More actions... Alt+Intro

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).

```
UD02_01_TextView.java × activity_ud0201_text_view.xml × strings.xml ×
Edit translations for all locales in the translations editor. Open editor

1 <resources>
2   <string name="app_name">PMDM_2021_2022</string>
3   <string name="texto1">Soy un botón</string>
4   <string name="hello_world">Hello World!!!</string>
5   <string name="texto_tv_java">Da igual lo que se escriba aquí, puesto que al lanzar la aplicación este texto se va a modificar</string>
6   <string name="texto_tv_html">Cadena html del tv_java. Se realizará en java</string>
7   <string name="texto_tv_string">String del tv_java. Se realizará en java</string>
8 </resources>
```

Uso recursos string

```
UD02_01_TextView.java × activity_ud0201_text_view.xml × strings.xml ×

15 android:text="Hello World!!!" />
16
17 <TextView
18     android:id="@+id/tv_java"
19     android:layout_width="wrap_content"
20     android:layout_height="wrap_content"
21     android:textColor="#00F"
22     android:text="@string/t" />
23     @string/texto_tv_html
24     @string/texto_tv_java
25 <View
26     android:layout="@string/texto1
27     android:layout="@string/texto_tv_string
28     android:background="@string/fab_transformation_scrim_behavior
29     @string/fab_transformation_sheet_behavior
30 <TextView
31     android:id="@+id/tv_html"
```

En el layout hacemos uso de recursos anteriores para todos los campos texto.

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

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

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

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

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

Ya esta todo correcto respecto a los textos.

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.

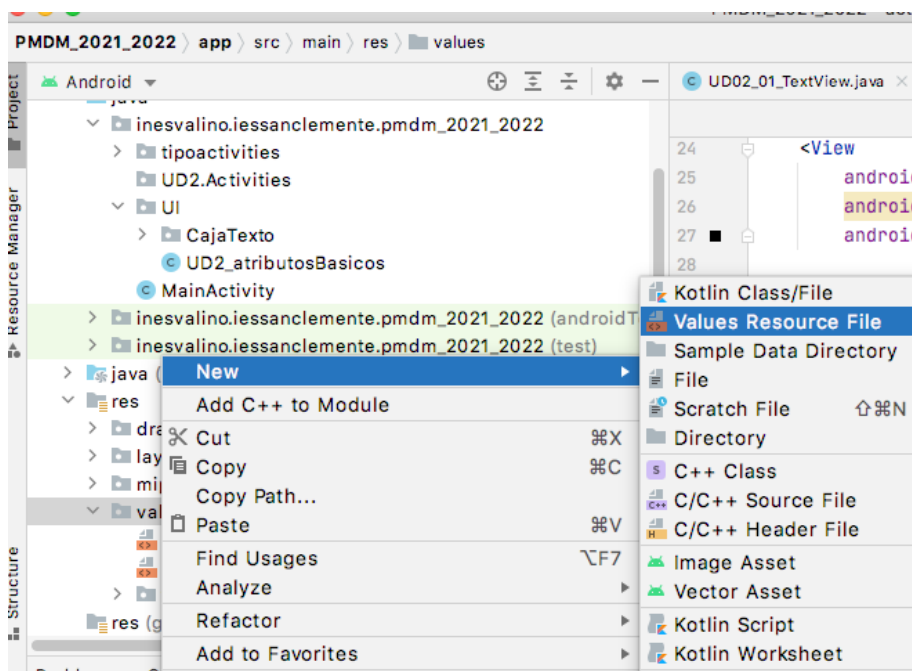
❖ A continuación se va a crear un fichero de recurso para los colores.

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).

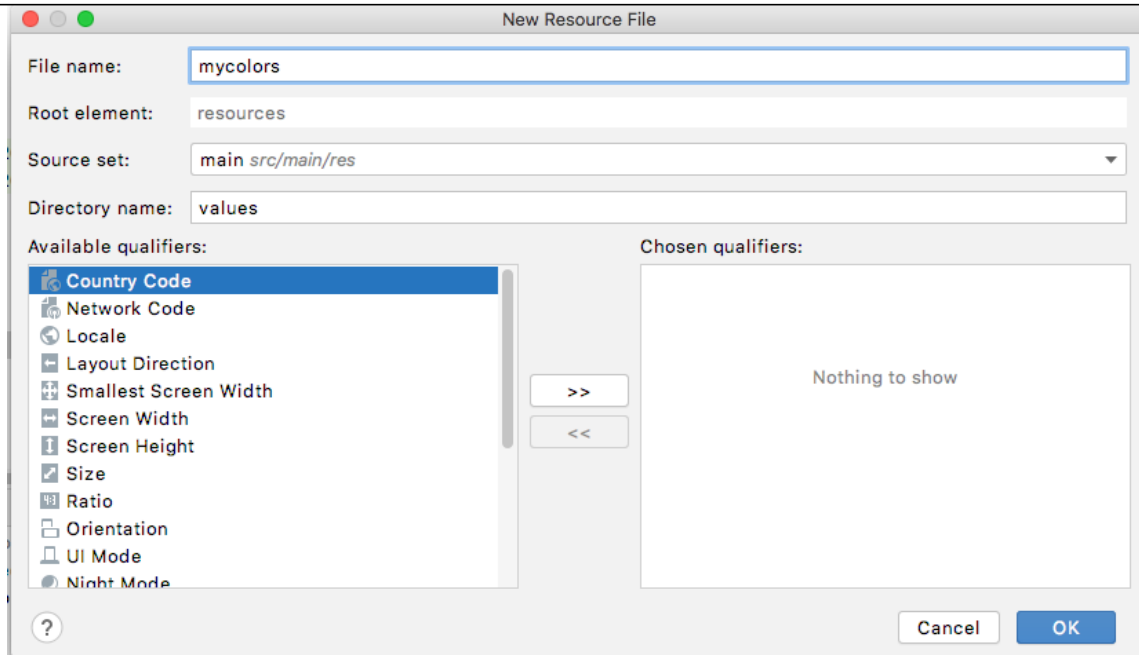
Normalmente haríamos uso de los que ya trae por defecto, porque de hecho crea por defecto un archivo colors.xml, pero vamos a crear uno nuevo para ver cuál sería el proceso de crear ficheros de recursos.

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**

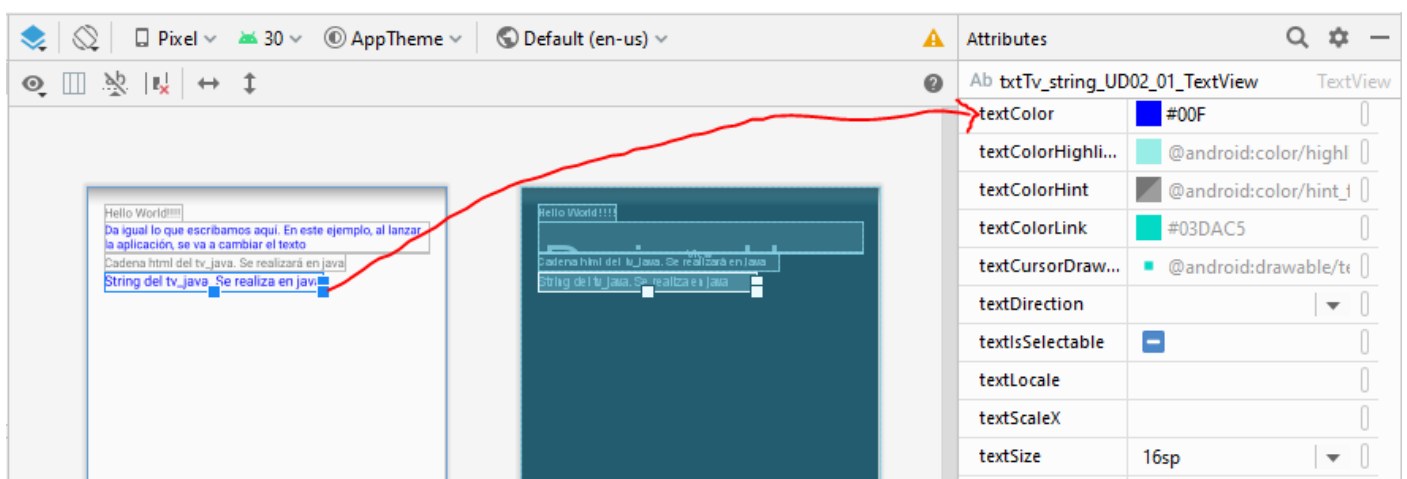


Editamos el archivo y agregamos los valores que queramos. En el ejemplo son colores, por lo que la etiqueta a utilizar es `<color>`. Todos los tipos que podemos usar en *res* (colores, dimensiones, cadena) tienen un atributo de nombre que es el nombre de la constante y un valor entre las etiquetas `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

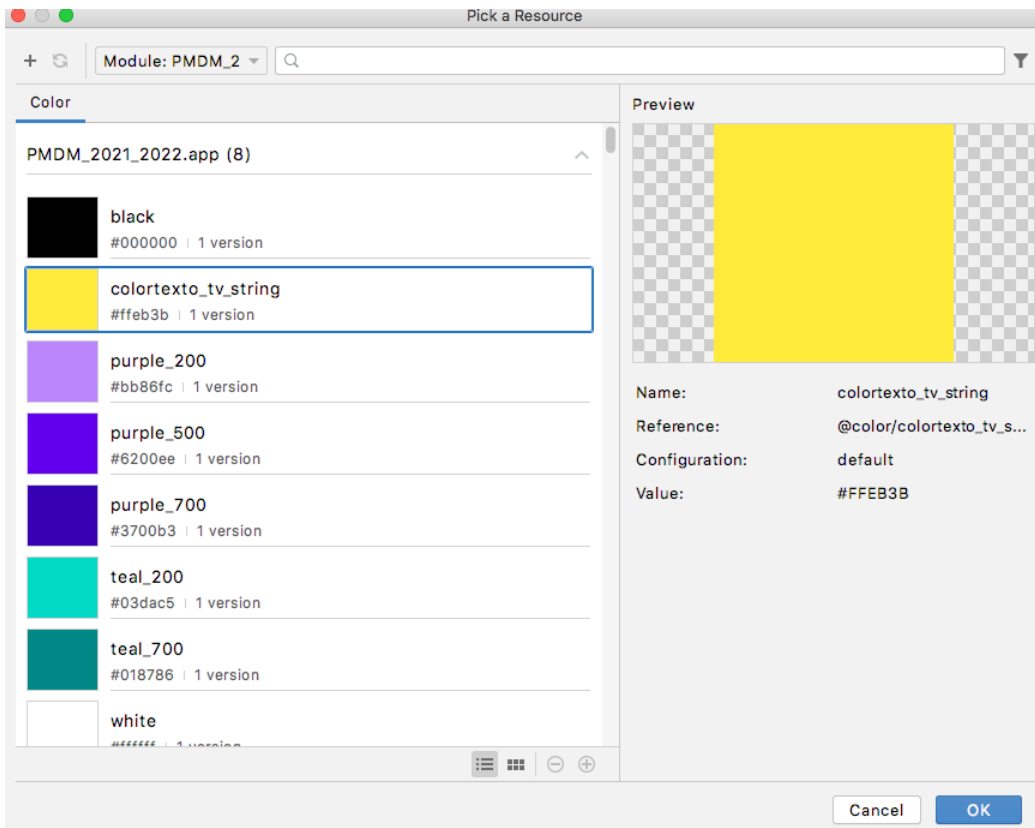
Creamos los colores que necesitamos, por ejemplo se crea el color pensando en darle un color a la última etiqueta:

```
<?xml version="1.0" encoding="Utf" ~/>AndroidStudioProjects/PMDM_Sar
</resources>
<resources>
  <color name="colortexto_tv_string">#FFEB3B</color>
</resources>
```

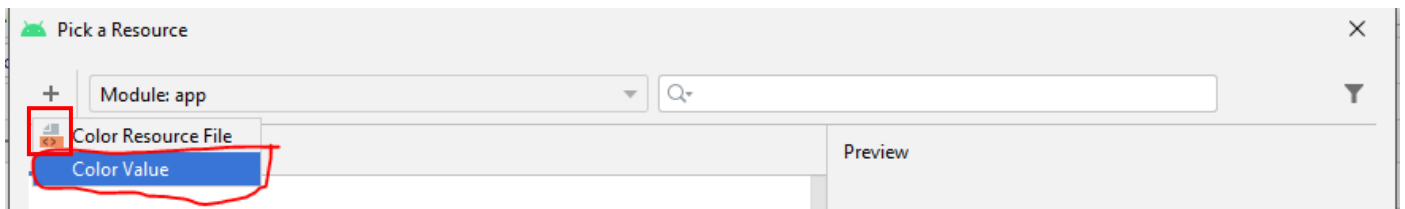
Ahora en el diseñador de layout seleccionamos la view a la que queremos asignar este color y hacemos clic en la barra (en la forma de elipse) de la derecha.



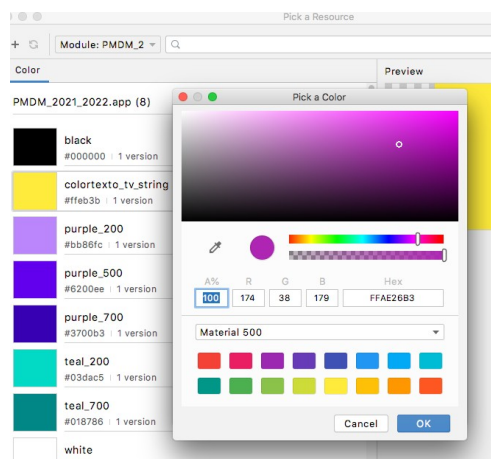
En la sección 'Color' a nivel de 'app', podemos seleccionar el color creado.



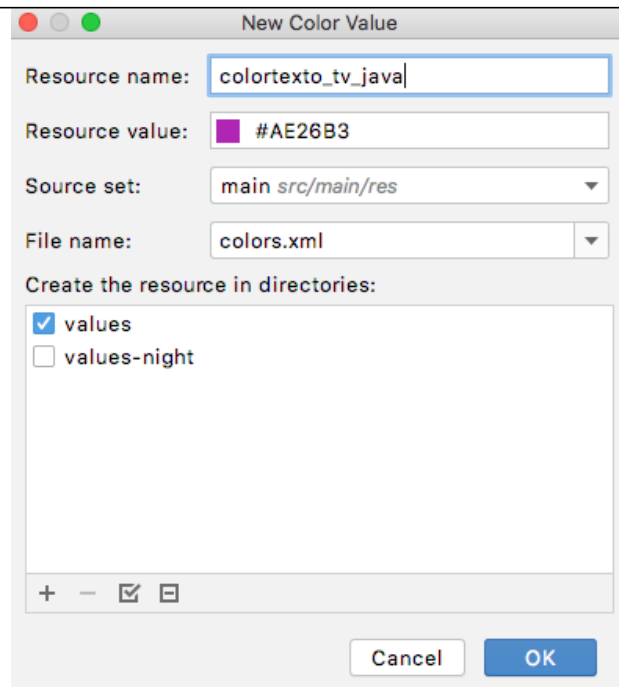
Clicamos en el signo + y seleccionamos 'Color Value'



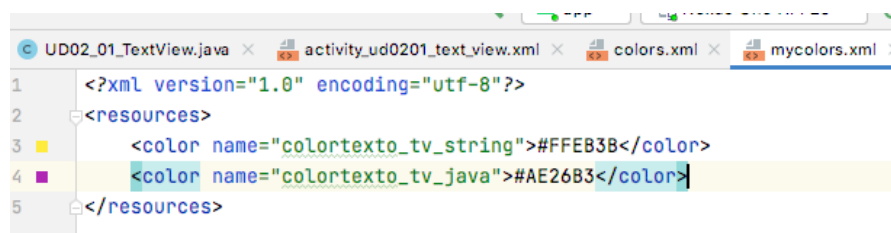
podríamos elegir el color que queremos usar, clicando sobre el símbolo Resource value: :



y asociarlo a un nombre:



El nuevo color se crea automáticamente en el fichero '*colors.xml*', podéis cortar y pegar esta línea en el fichero '*mycolors.xml*'



Podéis modificar el archivo de recursos de '*mycolors.xml*' añadiendo los colores que queráis para las otros texto.

- ❖ Para usar estos recursos en el layout, habrá que ir haciendo en todos los campos `textColor` como en la línea marcada :

```
<TextView
    android:id="@+id/tv_java"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@color/colortexto_tv_java"
    android:text="@string/texto_tv_java" />
```

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

- ❖ Otro recurso que se puede definir en constantes en un archivo XML son la dimensiones (tamaños). 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.

En nuestro ejemplo hay dos valores candidatos a definir en constantes de tipo *dimen*. Seguro que el estudiante podrá hacerlo. La etiqueta de las dimensiones es: `<dimen> </dimen>`. Recordad:

- utilizar el atributo `'name'`.
- el tamaño que se guarde en `<dimen>` tiene que especificar la unidad, *sp* ou *dp* de la 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 `linksclickables`. Por ejemplo:

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

Por último, si queremos mostrar enlaces html en el TextView se hace 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());
```

donde la línea 2 es quien lo hace. 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.

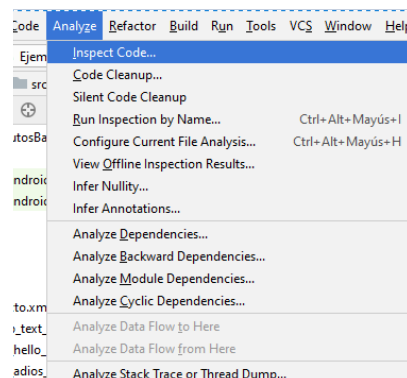
```
<TextView
    android:id="@+id/textView7"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Dime tu nombre:"
    android:textSize="18sp"
    android:labelFor="@id/etNome_UD03_01_Intents"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

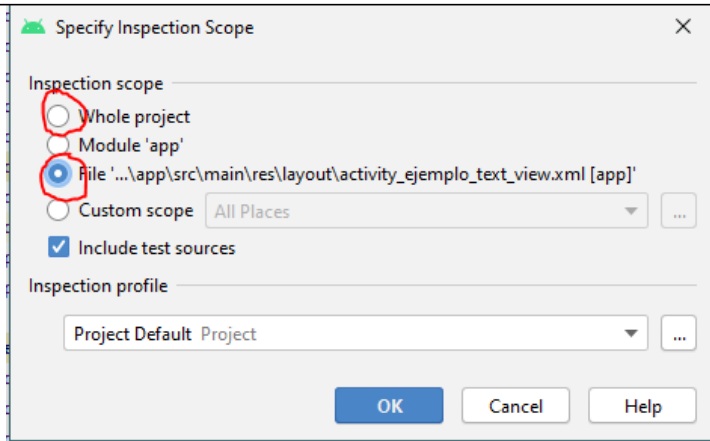
- ❖ Para comprobar que no tenemos problemas de accesibilidad:

Comprobando la accesibilidad

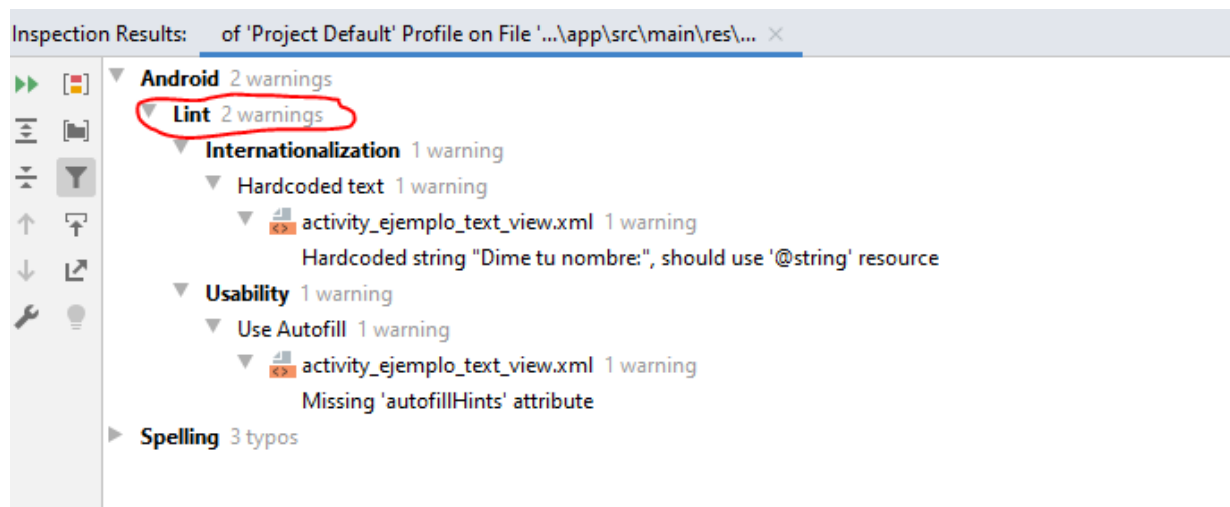
Vamos al menú de Android Studio y escogemos la opción **Analyze =>Inspect Code**.



Indicamos si queremos analizar el proyecto completo o la activity actual



por ejemplo, en la activity actual:



1.4. Empleando recursos definidos Res

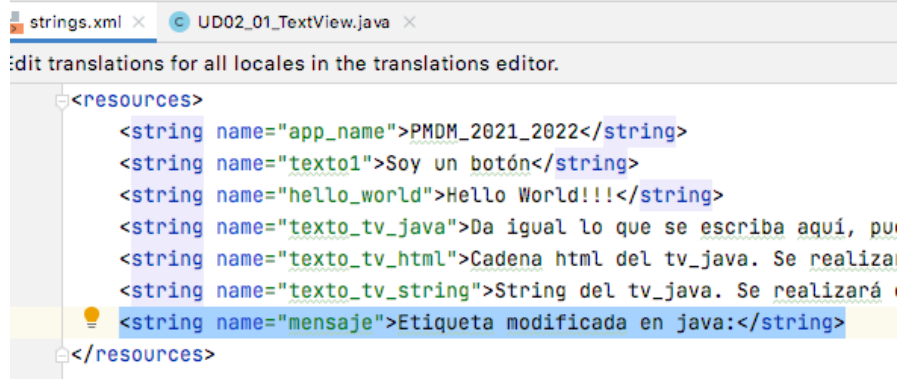
En el tema de internacionalización veremos cómo podemos tener cadenas en varios idiomas y que Android elija la correcta en función del idioma elegido a nivel de sistema operativo en los ajustes.

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, como por ejemplo:

```
tvJava.setText("Etiqueta modificada en java: "+tvOriginal.getText());
```

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:



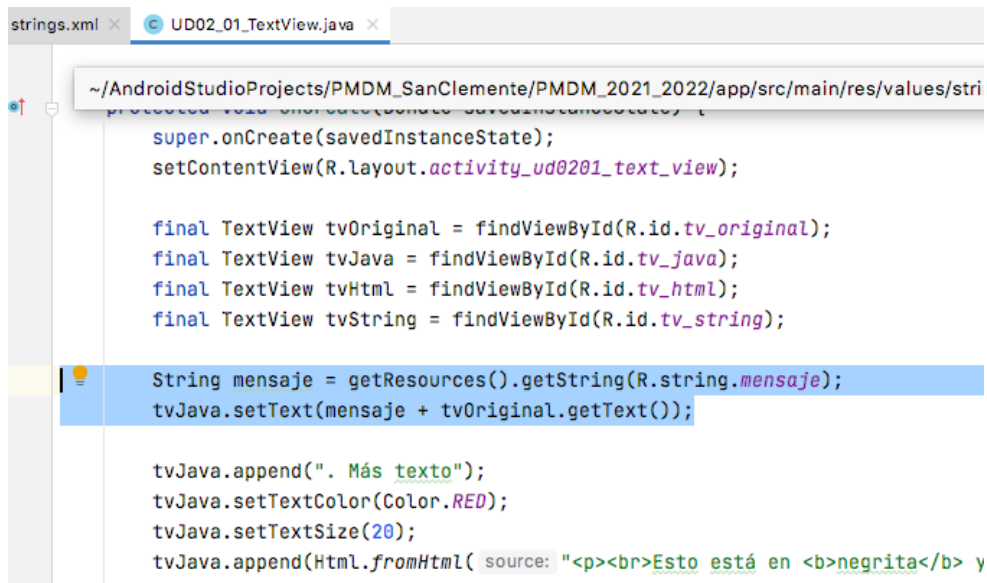
```

strings.xml x UD02_01_TextView.java x
edit translations for all locales in the translations editor.
<resources>
  <string name="app_name">PMDM_2021_2022</string>
  <string name="texto1">Soy un botón</string>
  <string name="hello_world">Hello World!!!</string>
  <string name="texto_tv_java">Da igual lo que se escriba aquí, pu
  <string name="texto_tv_html">Cadena html del tv_java. Se realiza
  <string name="texto_tv_string">String del tv_java. Se realizará
  <string name="mensaje">Etiqueta modificada en java:</string>
</resources>

```

Como vemos, el nombre de la constante es: *mensaje*

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



```

strings.xml x UD02_01_TextView.java x
~/AndroidStudioProjects/PMDM_SanClemente/PMDM_2021_2022/app/src/main/res/values/strings.xml
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_ud0201_text_view);

    final TextView tvOriginal = findViewById(R.id.tv_original);
    final TextView tvJava = findViewById(R.id.tv_java);
    final TextView tvHtml = findViewById(R.id.tv_html);
    final TextView tvString = findViewById(R.id.tv_string);

    String mensaje = getResources().getString(R.string.mensaje);
    tvJava.setText(mensaje + tvOriginal.getText());

    tvJava.append(". Más texto");
    tvJava.setTextColor(Color.RED);
    tvJava.setTextSize(20);
    tvJava.append(Html.fromHtml( source: "<p><br>Esto está en <b>negrita</b> y

```

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 mostrada en la imagen superior para obtener primero la cadena definida en el recurso o usar el método `append()`. Recuerdad 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>

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 que debes conocer en el manejo de TextViews

- Referenciar a un TextView con el método *findViewById*.
- Recuperar el contenido del texto.
- Cambiar el contenido del texto, pudiendo utilizar recursos guardados en /res /values.
- Agregar texto nuevo a uno existente.
- Modificar propiedades básicas, como color, tamaño, visibilidad, ... (y métodos getXXX 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 (como veremos en 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();
    }
});

```