

Práctica 1

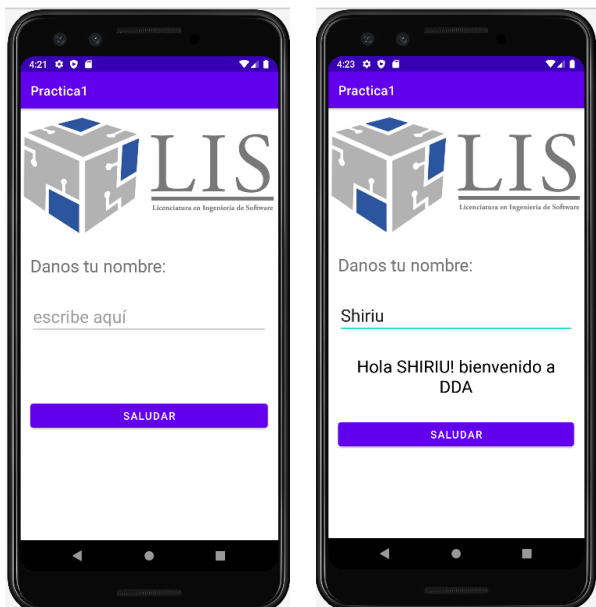
Contenido

LinearLayout	3
Lógica de la App y OnClickListener	8
Toast para informar al usuario	10
Recursos strings.....	11
Definiendo recurso string directamente en el archivo strings.xml	12
Extrayendo el recurso harcodeado en recurso string.....	13
Creación del recurso string desde el código.....	15
Soporte de varios idiomas.....	18

Objetivo: Utilizar el LinearLayout en una aplicación de Android con Java que incluya el uso de diferentes Views para la interacción básica del usuario con la aplicación.

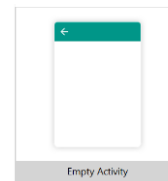
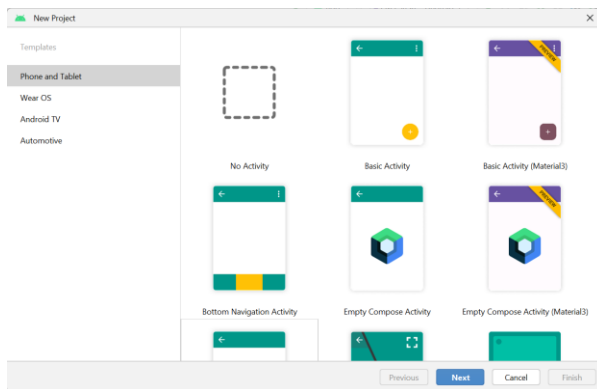
Funcionamiento: El usuario ingresa su nombre, la aplicación muestra un saludo

Resultado Esperado:

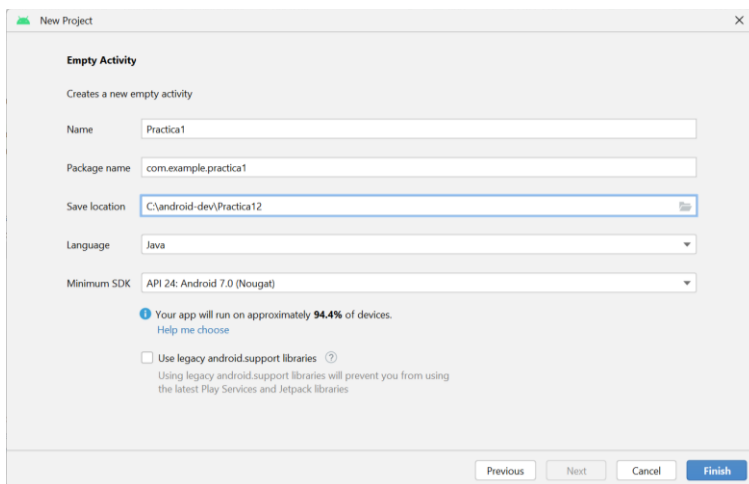


Crear un nuevo proyecto **File->New Project**

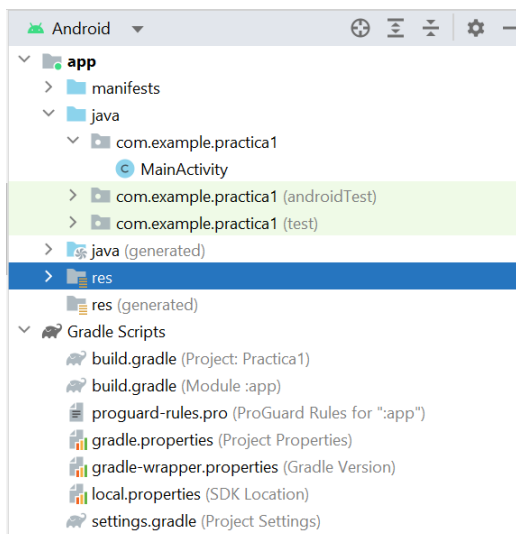
Seleccionamos **Empty Activity**



Llenamos los campos para definir nuestro proyecto.

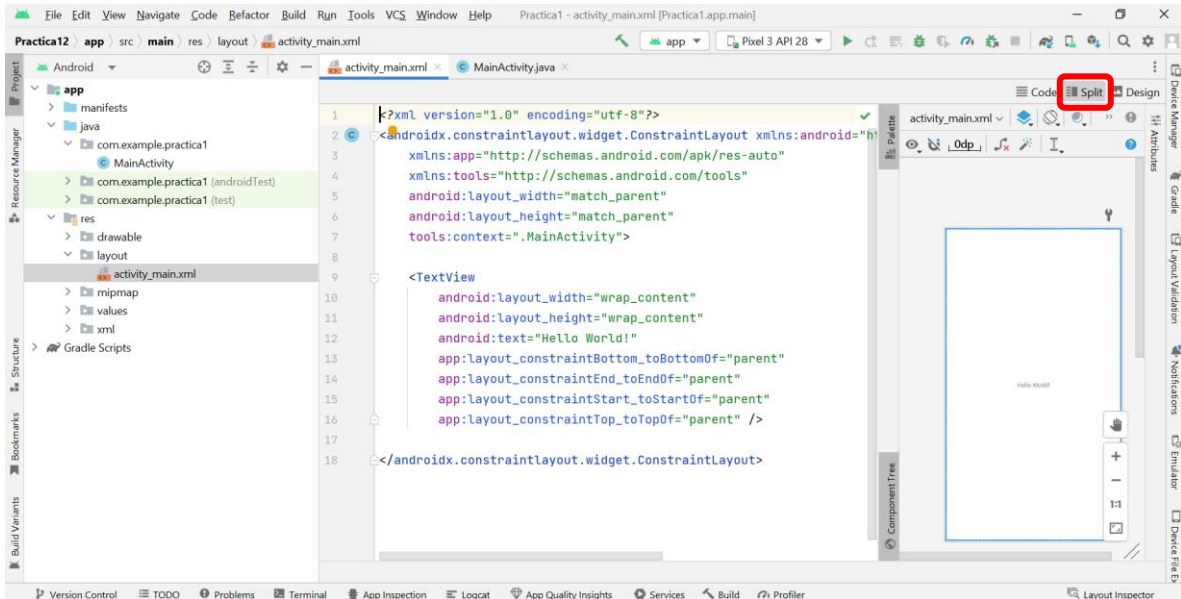


Después de pulsar el botón finalizar, se cargarán los elementos básicos para el funcionamiento de nuestra aplicación



LinearLayout

Nos dirigimos a la carpeta **res->layout->activity_main.xml** del proyecto, enseguida podremos ver la pantalla o interfaz de usuario en el IDE. Para poder visualizar el código en XML, seleccionamos el ícono **Split** ubicado en la parte superior derecha (ver recuadro rojo en la siguiente Figura)



Ahora sustituiremos la línea 2

(`<androidx.constraintlayout.widget.ConstraintLayout...>`) por (`<LinearLayout ...>`) quedando de la siguiente manera:

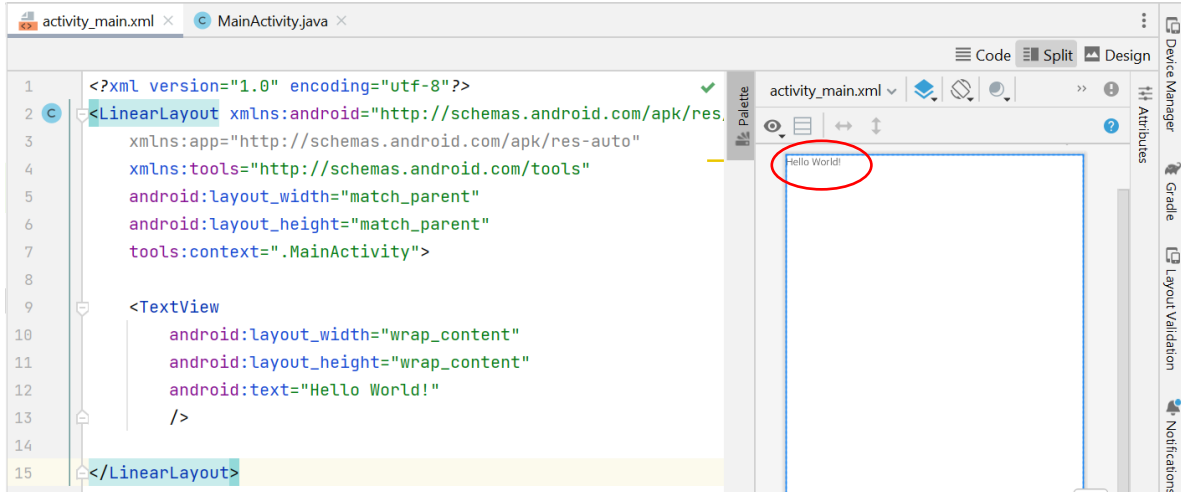
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</LinearLayout>
```

Las líneas tachadas y en gris del código anterior, se pueden eliminar, ya que pertenecen al **ConstraintLayout** que veremos en práctica posterior.

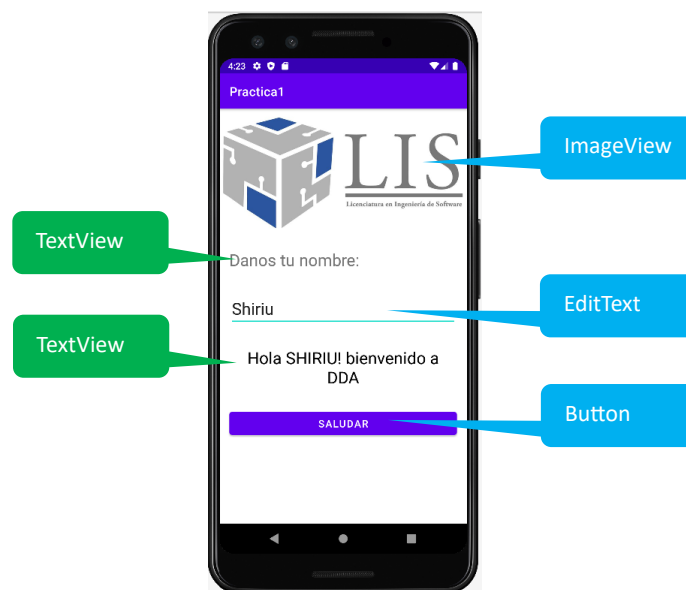
Por otra parte, podrás notar que ahora el texto de nuestro **TextView** cambió de la posición centrada en pantalla, a la parte superior izquierda



Todos los **Views** dentro de un **LinearLayout** se alinean de manera horizontal o vertical. Por ello, es importante incluir la propiedad `android:orientation` dentro de su etiqueta (tag)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

A continuación, ya podemos incluir todos los Views de nuestra aplicación



El siguiente código, contiene los TextViews, EditText y Button

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Danos tu nombre:"
        android:textSize="26sp"
        android:layout_margin="16sp"
    />

    <EditText
        android:id="@+id/user_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="escribe aquí"
        android:textSize="26sp"
        android:layout_margin="16sp"
    />

    <TextView
        android:id="@+id/txt_saludo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@color/black"
        android:gravity="center"
        android:textSize="26sp"
        android:layout_margin="16sp"
    />

    <Button
        android:id="@+id/btn_saludar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:layout_margin="16sp"
        android:text="Saludar"
    />

</LinearLayout>
```

wrap_content: ajusta el objeto al tamaño de su contenido

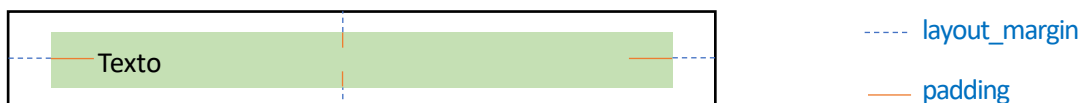
Match_parent: ajusta el objeto al tamaño del Layout

gravity: permite alinear el contenido de un elemento en diferentes posiciones

layout_margin: agrega un margen entre el Layout y el objeto

Otra propiedad importante para ajustar los elementos en la pantalla, es el **android:padding** el cual agrega un margen entre el contenido del objeto y sus bordes.

La siguiente Figura muestra la diferencia entre el **layout_margin** y **padding**.

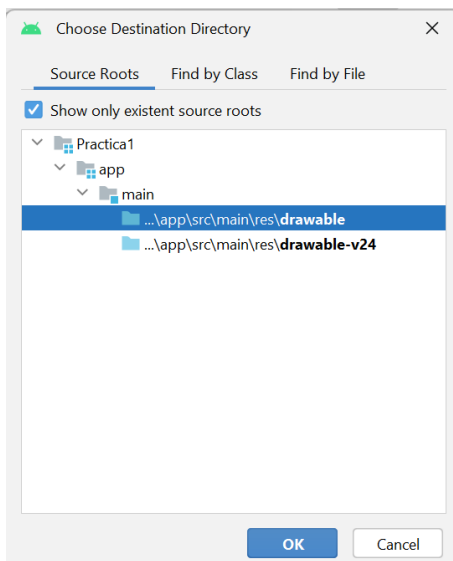


El resultado hasta el momento es el siguiente:

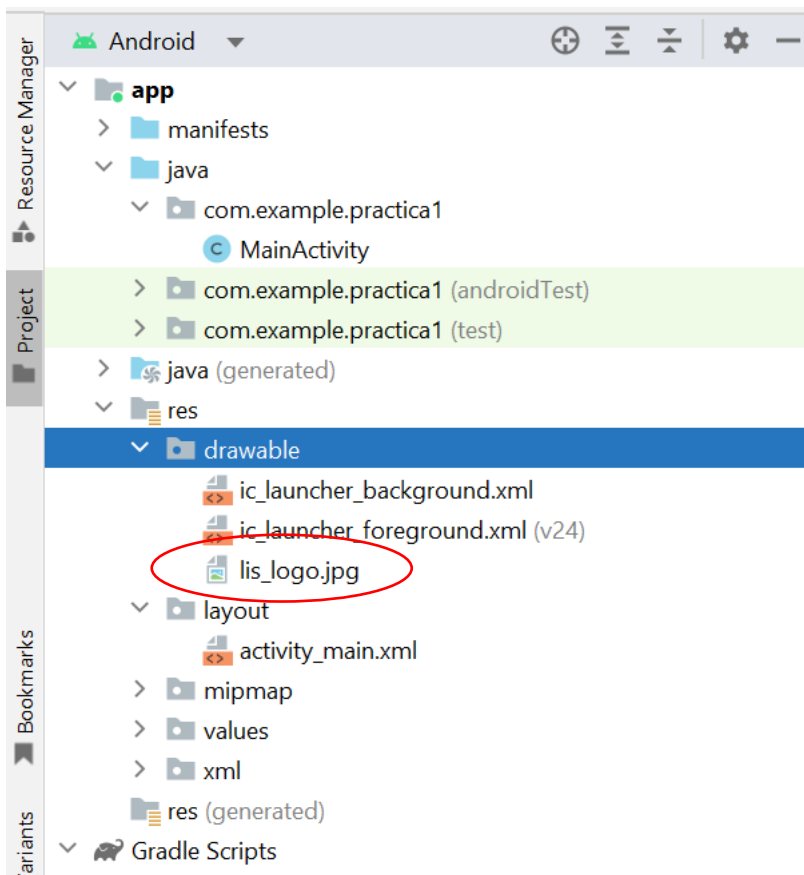


Ahora incluiremos la imagen con el logo de la LIS. Para ello, primero descargamos la imagen la guardamos en cualquier parte de nuestro PC y posterior lo pegamos en la carpeta **drawable** de nuestro proyecto.

El IDE nos preguntará donde lo queremos guardar, seleccionamos **drawable** y pulsamos OK.



A continuación, podremos ver que la imagen ya se incluyó en el proyecto:



Ahora incluimos el **ImageView**

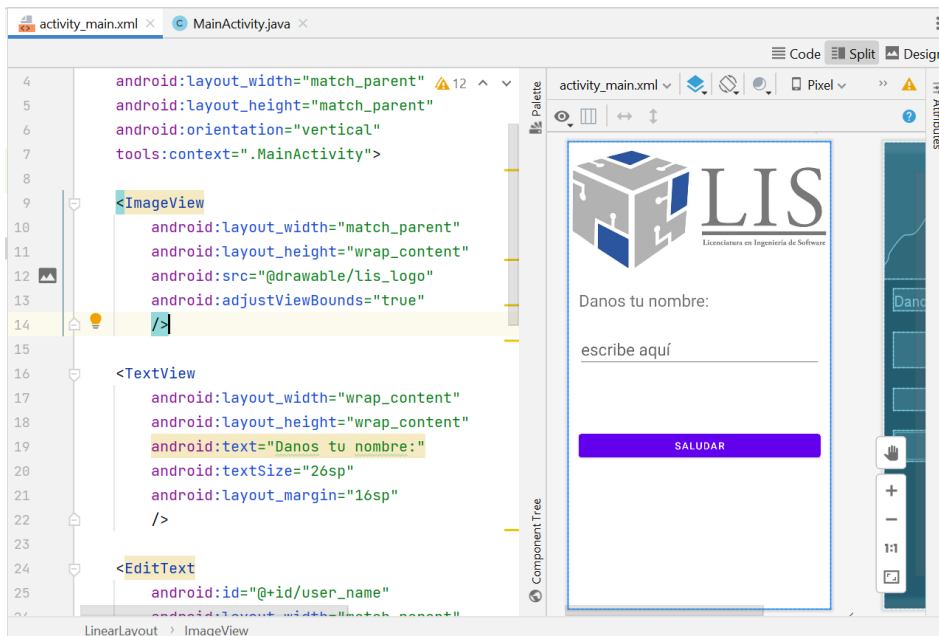
```
<ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/lis_logo"
    android:adjustViewBounds="true"
/>
```

src permite definir un recurso, en nuestro caso en la carpeta **drawable/<nombre_recurso>**

Es importante que el nombre del recurso (el nombre de nuestra imagen) esté escrita en minúsculas.

adjustViewBounds evita que la imagen incluya márgenes internos y se ajusta al **ImageView**

El resultado es el siguiente:



Lógica de la App y OnclickListener

Para agregar la lógica de nuestra aplicación, nos dirigimos al **MainActivity.java**, el cual se encuentra en la carpeta **app/java/com.example.practica1**



Dentro del método onCreate definiremos tres objetos con los que interactuará el usuario:

1. El campo EditText: donde el usuario ingresará su nombre
2. El TextView: donde la aplicación mostrará el mensaje
3. El Button: el cual pulsará el usuario para ejecutar una acción

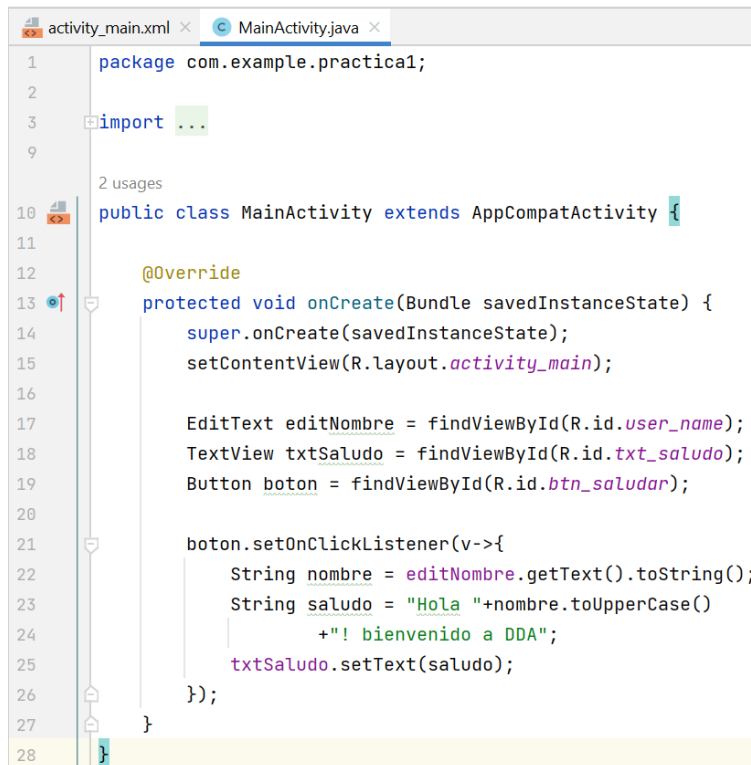
```
EditText editNombre = findViewById(R.id.user_name);
TextView txtSaludo = findViewById(R.id.txt_saludo);
Button boton = findViewById(R.id.btn_saludar);
```

El método **findViewById()** regresa el objeto definido en el **activity_main.xml**

Para desencadenar la acción cuando el usuario pulse el botón; será necesario llamar el método **setOnClickListener** el cual recibe como argumento una función lambda en la cual se aplica la lógica de la aplicación. En nuestro ejemplo, sólo se genera un string con un saludo al usuario.

```
boton.setOnClickListener(v->{
    String nombre = editNombre.getText().toString();
    String saludo = "Hola "+nombre.toUpperCase()
        + "! bienvenido a DDA";
    txtSaludo.setText(saludo);
});
```

Una vez leído el texto ingresado por el usuario, se crea un String con el saludo y finalmente se asigna el mensaje al TextView para mostrarlo en pantalla.



```
1 package com.example.practical;
2
3 import ...
4
5
6
7
8
9
10 public class MainActivity extends AppCompatActivity {
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16
17         EditText editNombre = findViewById(R.id.user_name);
18         TextView txtSaludo = findViewById(R.id.txt_saludo);
19         Button boton = findViewById(R.id.btn_saludar);
20
21         boton.setOnClickListener(v->{
22             String nombre = editNombre.getText().toString();
23             String saludo = "Hola "+nombre.toUpperCase()
24                 + "! bienvenido a DDA";
25             txtSaludo.setText(saludo);
26         });
27     }
28 }
```

El resultado final de la aplicación, después de que el usuario escribe su nombre y pulsa el botón “SALUDAR”, es el siguiente:



Toast para informar al usuario

Hasta este punto, la lógica de la aplicación no valida si el usuario no ingresa su nombre. Para estos casos, podemos utilizar el componente Toast para enviar una notificación emergente al usuario.

El Toast es un objeto de vista que se despliega como un elemento emergente en la interfaz del usuario, con el fin de mostrar un mensaje. Para declararlo, se debe agregar un contexto, el mensaje y la duración.

Ejemplo:

```
Toast.makeText(this, "Debes ingresar tu nombre", Toast.LENGTH_SHORT).show();
```

En nuestro caso, lo podríamos incluir dentro del OnClickListener

```
boton.setOnClickListener(v->{
    String nombre = editNombre.getText().toString();

    if(!nombre.isEmpty()) {
        String saludo = "Hola " + nombre.toUpperCase()
            + "! bienvenido a DDA";
        txtSaludo.setText(saludo);
    }else{
        Toast.makeText(this, "Debes ingresar tu nombre",
            Toast.LENGTH_SHORT).show();
    }
});
```

El resultado de esta validación se observa en la siguiente Figura:

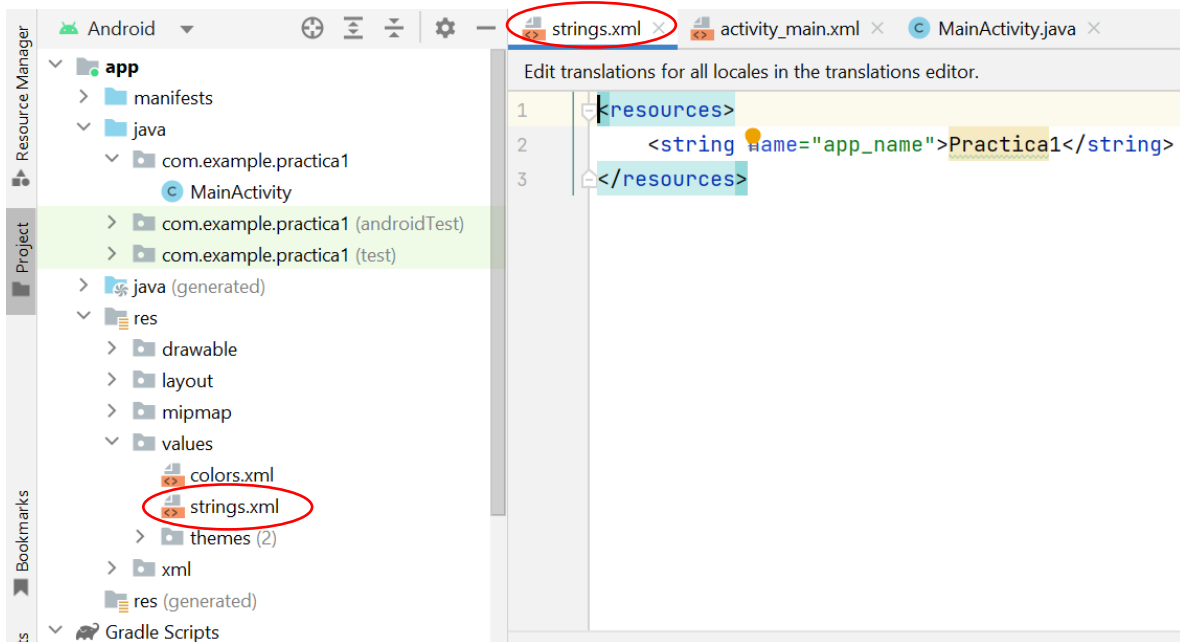


Recursos strings

Hasta este punto, nuestro código tiene incrustado texto directo en el código, es decir, nuestro código está “hardcodeado”, lo cual no es una buena práctica.

Por ello, podemos hacer uso de los recursos strings para crear etiquetas reutilizables en nuestra aplicación.

Para acceder a ellos, nos vamos a la carpeta **res/values/** de nuestro proyecto. Ahí encontraremos el archivo **string.xml**



A continuación, pondremos nuestras etiquetas de la aplicación. Se recomienda que las etiquetas se escriban en el idioma inglés, siguiendo la estructura:

```
<string name="nombre_etiqueta">Label Name</string>
```

La convención en Android es que los nombres de las etiquetas se escriben en minúsculas y las palabras separadas con guion bajo.

Existen tres formas de definir los recursos strings:

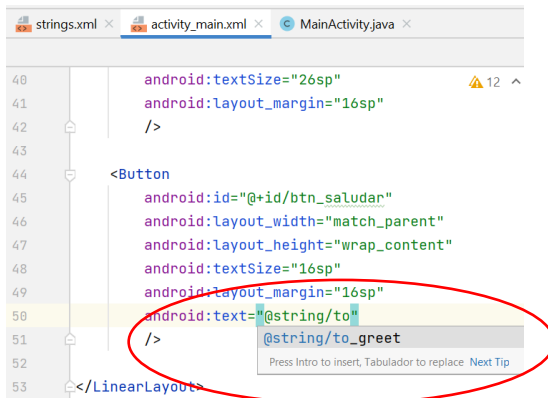
1. Definiendo recurso directamente en el archivo strings.xml
2. Extrayendo el recurso harcodeado en recurso string
3. Creación del recurso string desde el código

Definiendo recurso string directamente en el archivo strings.xml

Consiste en escribir la estructura de los recursos string directamente. El siguiente ejemplo muestra la definición del string para el texto que tiene el botón Saludar de nuestra aplicación.

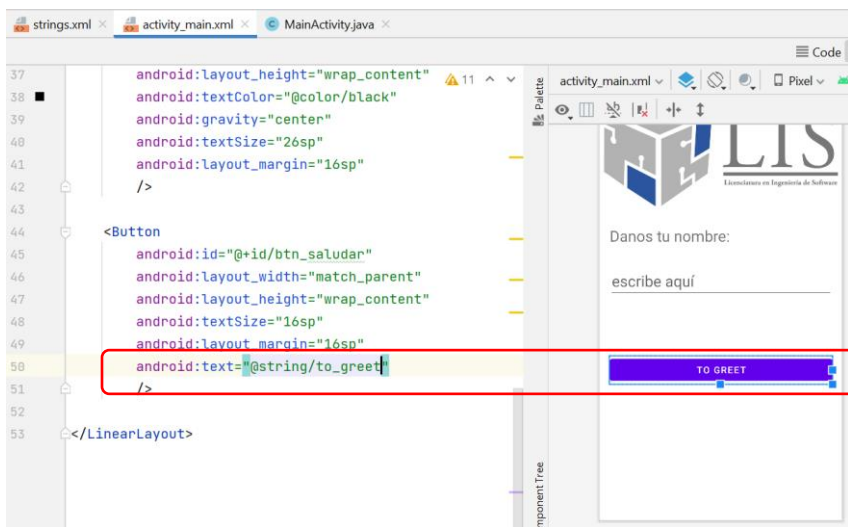
```
<resources>
  <string name="app_name">Practical</string>
  <string name="to_greet">To Greet</string>
</resources>
```

Ahora en el archivo **activity_main.xml** reemplazaremos el texto harcodeado con el recurso string.



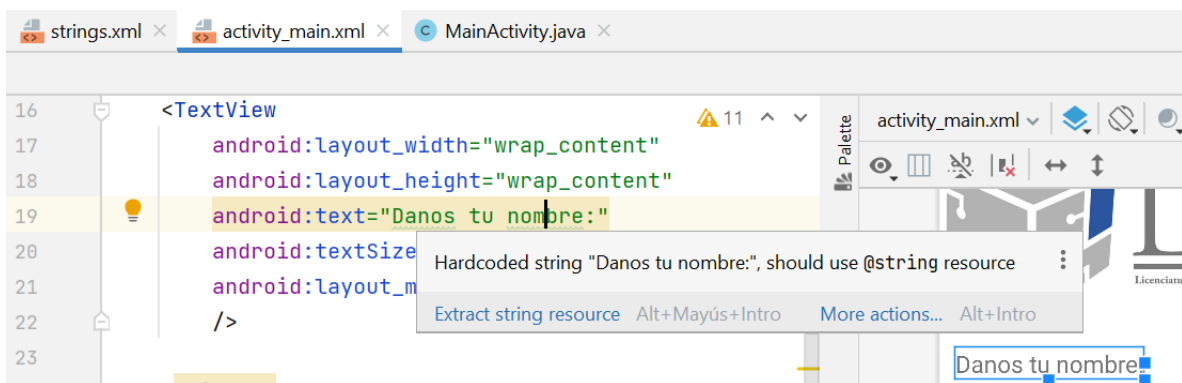
```
40         android:textSize="26sp"
41         android:layout_margin="16sp"
42     />
43
44     <Button
45         android:id="@+id/btn_saludar"
46         android:layout_width="match_parent"
47         android:layout_height="wrap_content"
48         android:textSize="16sp"
49         android:layout_margin="16sp"
50         android:text="@string/to_greet"
51     />
52
53 </LinearLayout>
```

Resultado:

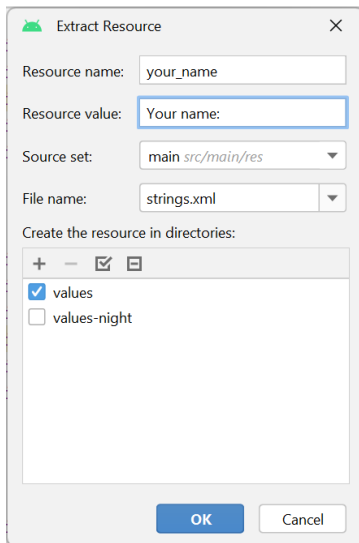


Extrayendo el recurso harcodeado en recurso string

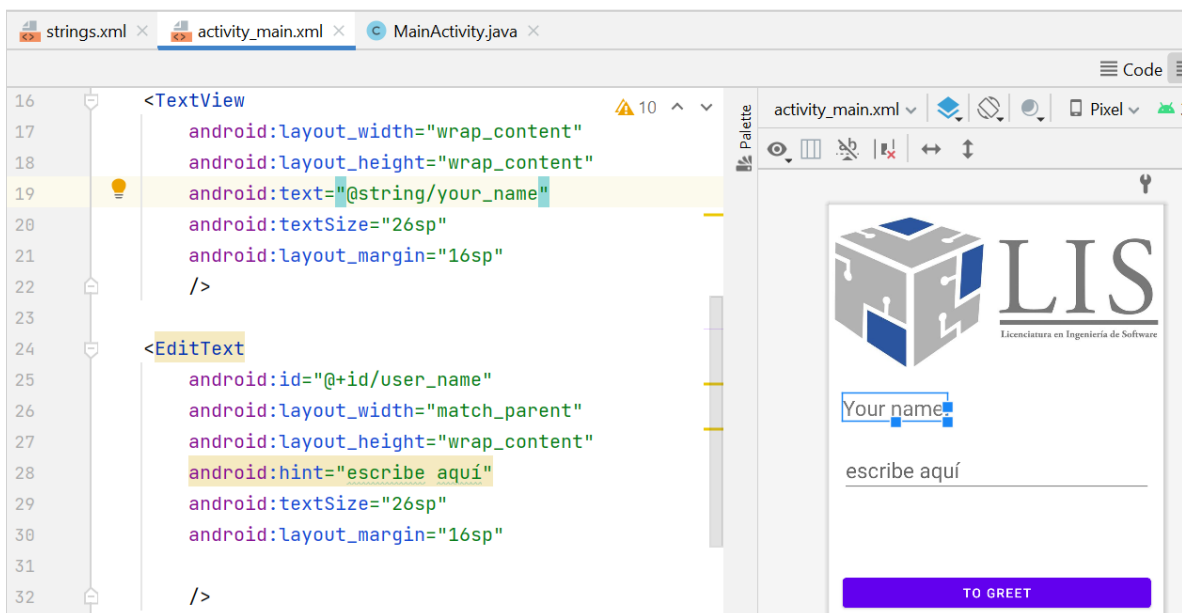
Primero debes posicionar el puntero del ratón sobre el texto harcodeado y enseguida se abrirá un menú contextual.



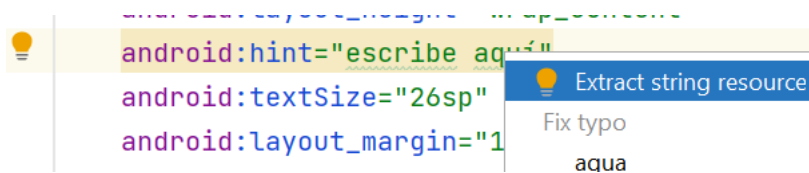
Con el menú contextual desplegado, pulsar las teclas Alt+Shift+Enter y se mostrará un formulario, en el que se definirá el nombre y valor del recurso. En mi caso, como el texto estaba en idioma español, tuve que traducir manualmente, tanto el nombre como el valor del recurso.



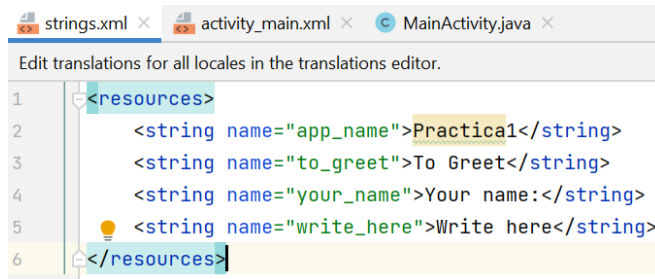
Al pulsar el botón OK, nos quedará de la siguiente forma:



En el caso de no utilizar el puntero del ratón, entonces posicionar el cursor en el texto harcodeado y pulsar las teclas Alt+BloqMayus+Enter, esto desplegará un menú contextual, del cual seleccionaremos la primera opción



El proceso es el mismo que ya se explicó, nuestro archivo strings.xml deberá estar de la siguiente manera:

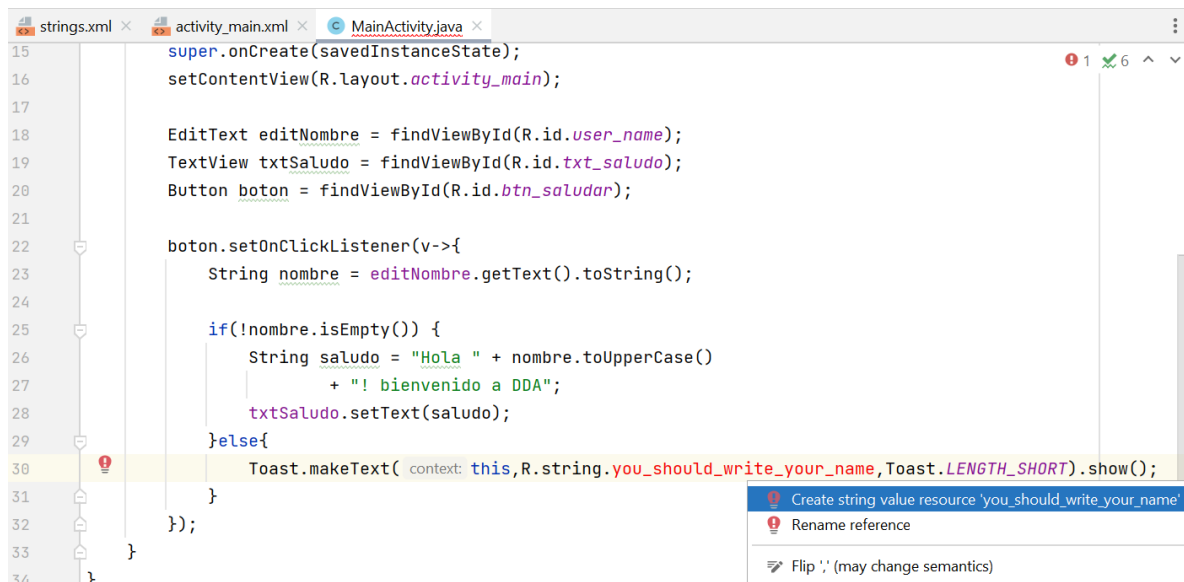


```
1 <resources>
2   <string name="app_name">Practica1</string>
3   <string name="to_greet">To Greet</string>
4   <string name="your_name">Your name:</string>
5   <string name="write_here">Write here</string>
6 </resources>
```

Creación del recurso string desde el código

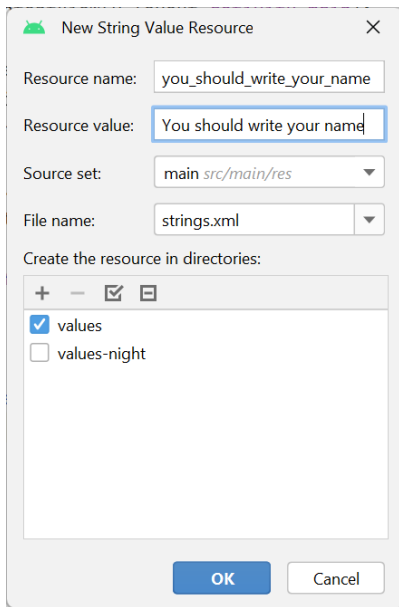
Esta opción es conveniente cuando aún no hemos harcodeado texto en nuestro código, por lo que podemos escribir el nombre de recurso string.

Por ejemplo, en el Toast tenemos un mensaje para el usuario. Lo borraremos y escribiremos en su lugar **R.string.nombre_recurso**. Como aún no existe, nos aparecerá en color rojo. Ponemos el cursor sobre el texto en rojo y pulsamos **Alt+BloqMayus+Enter** lo cual nos desplegará un menú contextual, y seleccionamos la primera opción.



```
15 super.onCreate(savedInstanceState);
16 setContentView(R.layout.activity_main);
17
18 EditText editNombre = findViewById(R.id.user_name);
19 TextView txtSaludo = findViewById(R.id.txt_saludo);
20 Button boton = findViewById(R.id.btn_saludar);
21
22 boton.setOnClickListener(v->{
23     String nombre = editNombre.getText().toString();
24
25     if(!nombre.isEmpty()) {
26         String saludo = "Hola " + nombre.toUpperCase()
27             + "! bienvenido a DDA";
28         txtSaludo.setText(saludo);
29     }else{
30         Toast.makeText( context: this,R.string.you_should_write_your_name,Toast.LENGTH_SHORT).show();
31     }
32 });
33 }
34 }
```

Enseguida nos desplegará un formulario con el nombre del recurso que acabamos de escribir, sólo tendremos que escribir el valor del recurso, en nuestro ejemplo es el mensaje que le queremos mostrar al usuario mediante el Toast



Finalmente, nos queda el código sin el error



Como podemos observar en el código anterior, tenemos un texto harcodeado contenido en String saludo. Sin embargo, este texto concatena el nombre que el usuario escriba en el EditText. Para utilizar un recurso string vamos a crearlo en el archivo strings.xml

```
<string name="welcome_message">Hi %s ! welcome to DDA course</string>
```

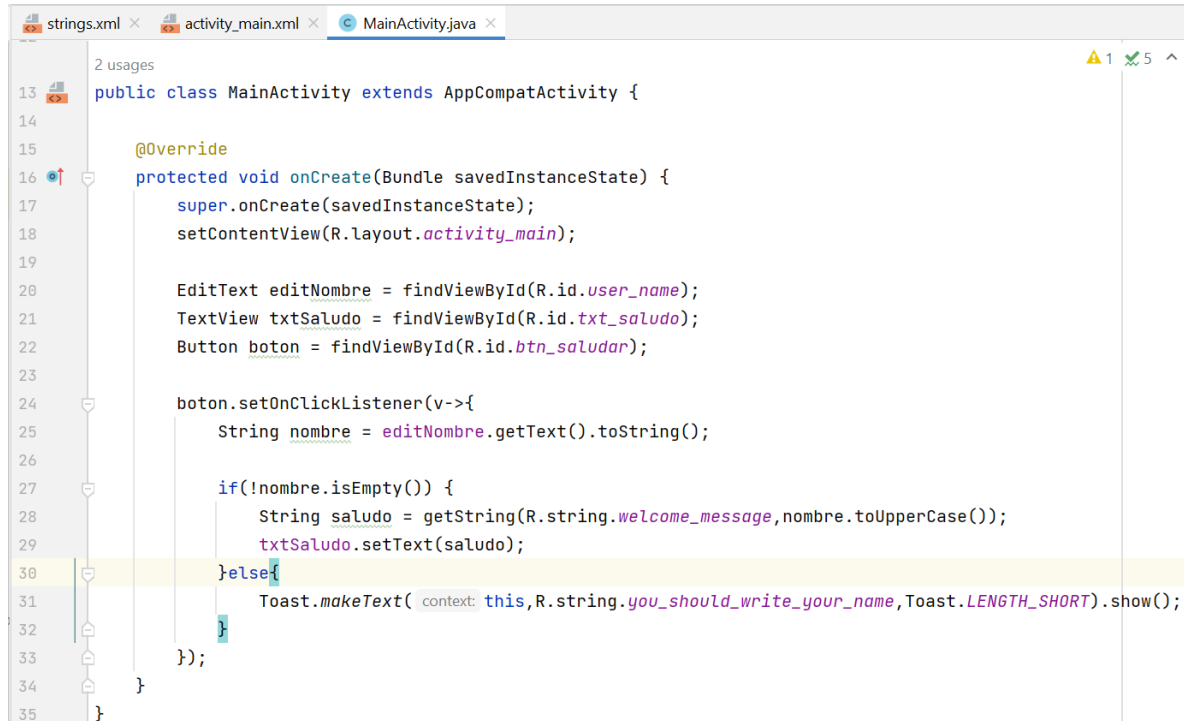

El signo %s indica que el texto va a recibir un String. En el caso de necesitar concatenar un valor numérico se usaría %d

Finalmente, incluimos nuestro recurso string en el código

```
String saludo = getString(R.string.welcome_message,nombre.toUpperCase());
```

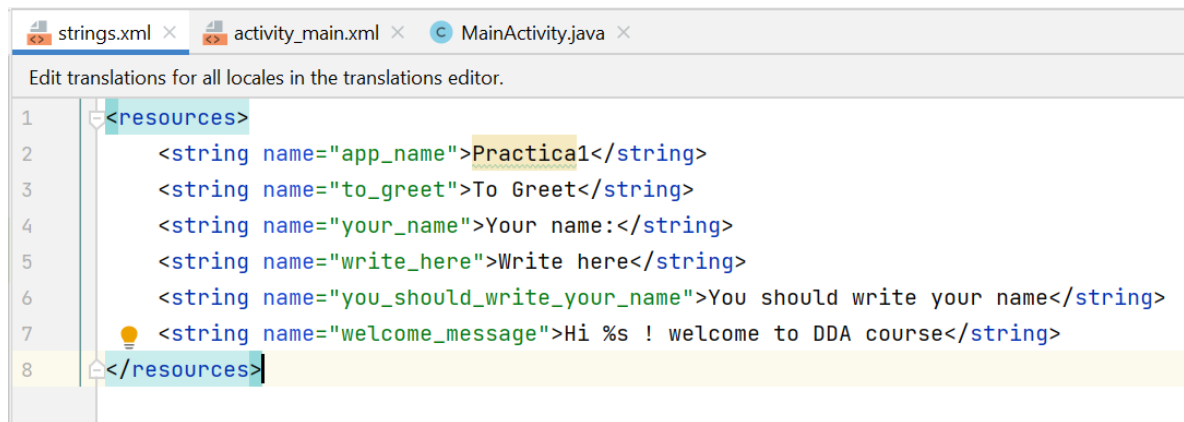
El método getString() recibe el formato del texto y los argumentos.

Código completo:

A screenshot of an IDE showing the MainActivity.java file. The code defines a MainActivity class that extends AppCompatActivity. It overrides the onCreate method, which calls super.onCreate, sets the content view to R.layout.activity_main, and finds three UI elements: an EditText for the user's name, a TextView for the greeting, and a Button for the 'saludar' action. The Button's onClick listener checks if the name field is empty. If not empty, it retrieves the greeting string using getString(R.string.welcome_message, nombre.toUpperCase()) and sets it on the TextView. If the field is empty, it shows a toast message 'you should write your name'.

```
13 public class MainActivity extends AppCompatActivity {
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19
20         EditText editNombre = findViewById(R.id.user_name);
21         TextView txtSaludo = findViewById(R.id.txt_saludo);
22         Button boton = findViewById(R.id.btn_saludar);
23
24         boton.setOnClickListener(v->{
25             String nombre = editNombre.getText().toString();
26
27             if(!nombre.isEmpty()) {
28                 String saludo = getString(R.string.welcome_message,nombre.toUpperCase());
29                 txtSaludo.setText(saludo);
30             }else{
31                 Toast.makeText( context: this,R.string.you_should_write_your_name,Toast.LENGTH_SHORT).show();
32             }
33         });
34     }
35 }
```

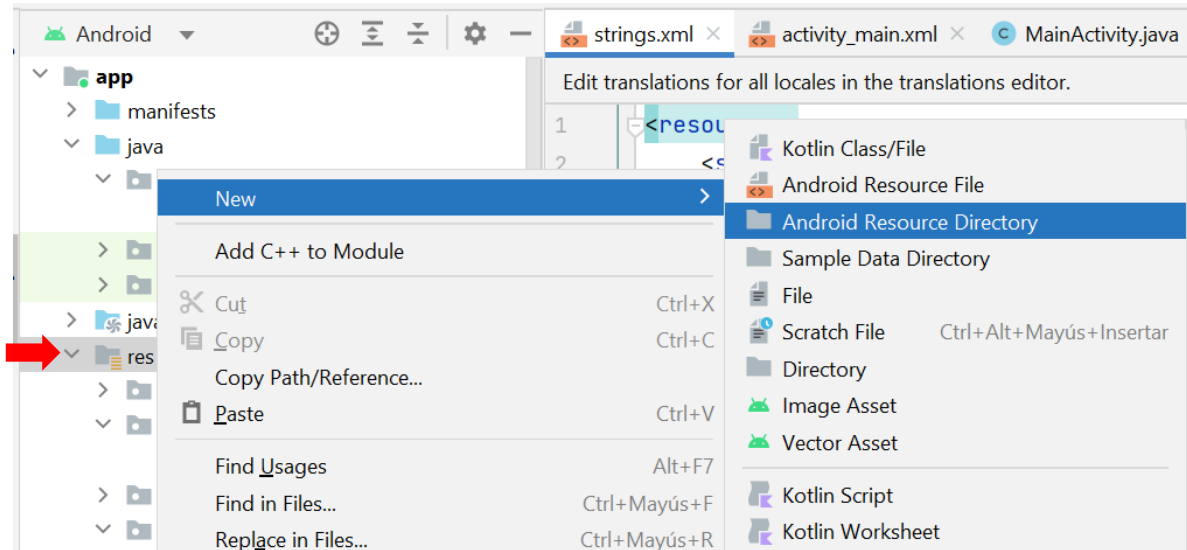
Recursos strings finales:

A screenshot of the strings.xml file in an IDE. The file contains several string resources. The last one, 'welcome_message', includes a lightbulb icon indicating a warning or suggestion, and the text 'Hi %s ! welcome to DDA course'.

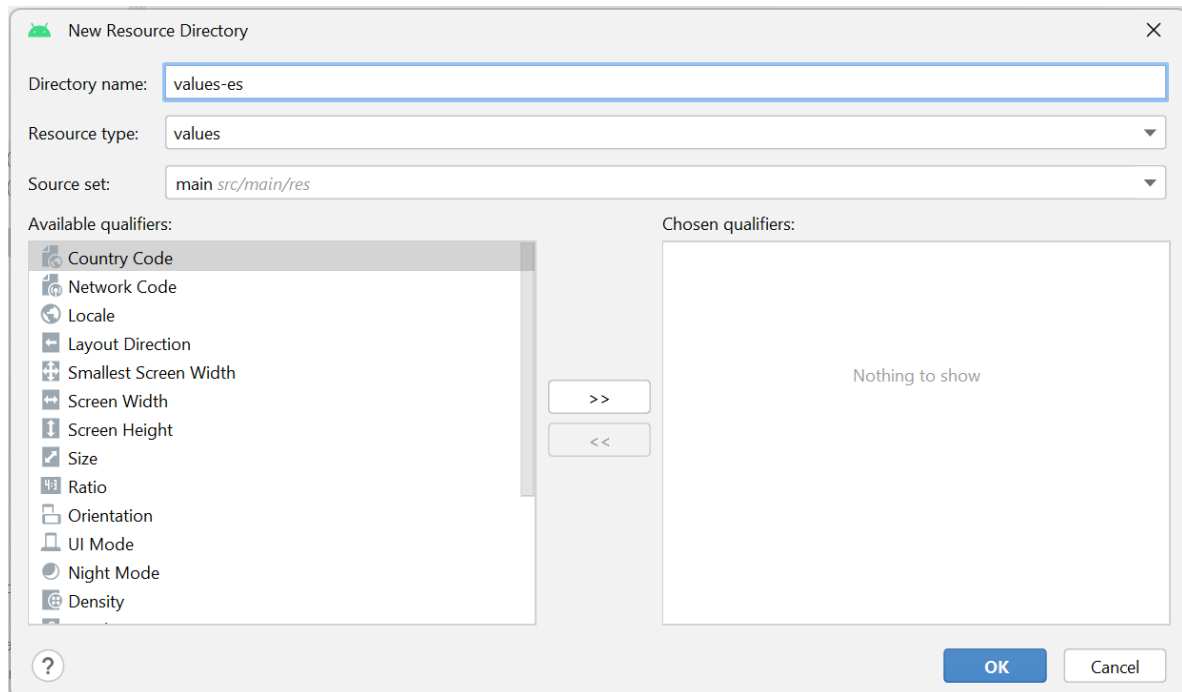
```
1 <resources>
2     <string name="app_name">Practica1</string>
3     <string name="to_greet">To Greet</string>
4     <string name="your_name">Your name:</string>
5     <string name="write_here">Write here</string>
6     <string name="you_should_write_your_name">You should write your name</string>
7     <string name="welcome_message">Hi %s ! welcome to DDA course</string>
8 </resources>
```

Soporte de varios idiomas

Gracias a los recursos strings podemos generar un archivo con diferentes idiomas. Para ello, daremos click derecho sobre la carpeta res de nuestro proyecto. Del menú contextual desplegado, seleccionaremos New->Android Resource Directory



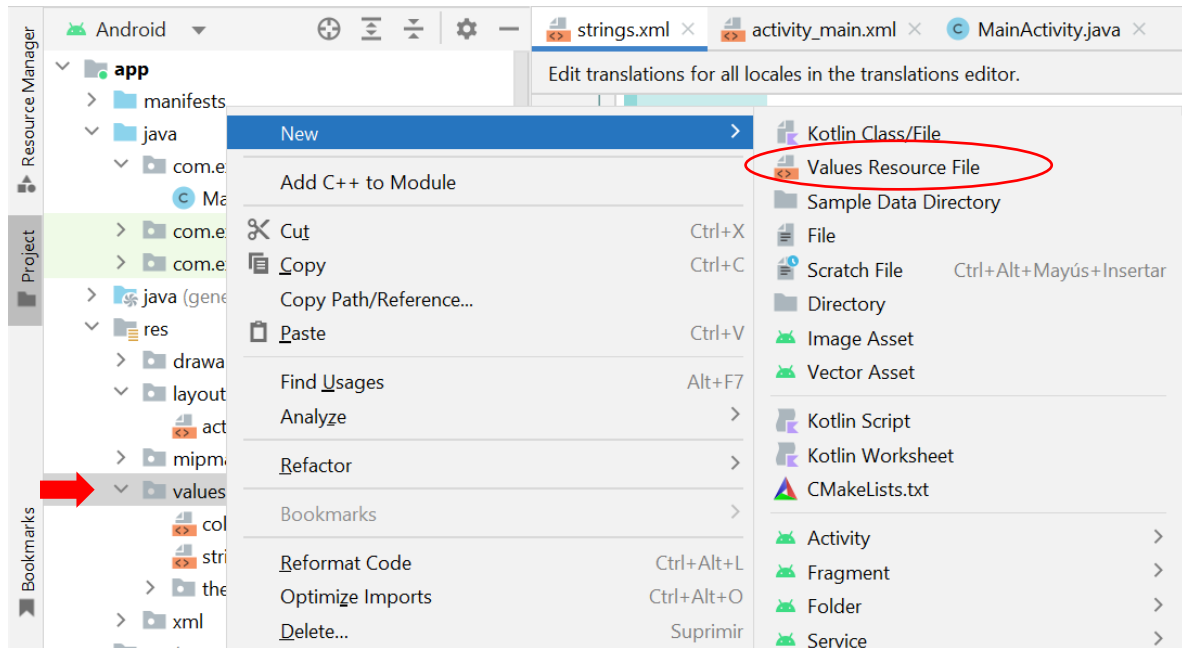
Nos desplegará un formulario y en el campo “Directory name:” lo nombraremos como values-es



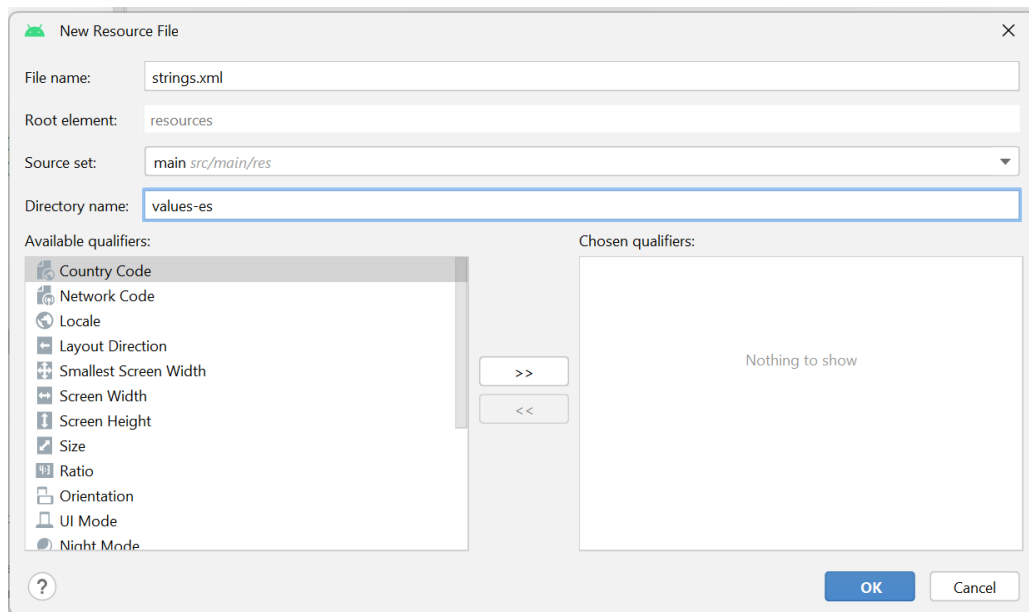
El sufijo “-es” obedece al estándar ISO 639-1 codes, correspondiente a los códigos de idiomas¹.

Al pulsar el botón OK, se generará una carpeta values-es dentro de res. La cual será accedida si la configuración del dispositivo está en español.

Hecho esto, iremos a la carpeta res/values y pulsaremos click derecho y seleccionamos New->Values Resource File

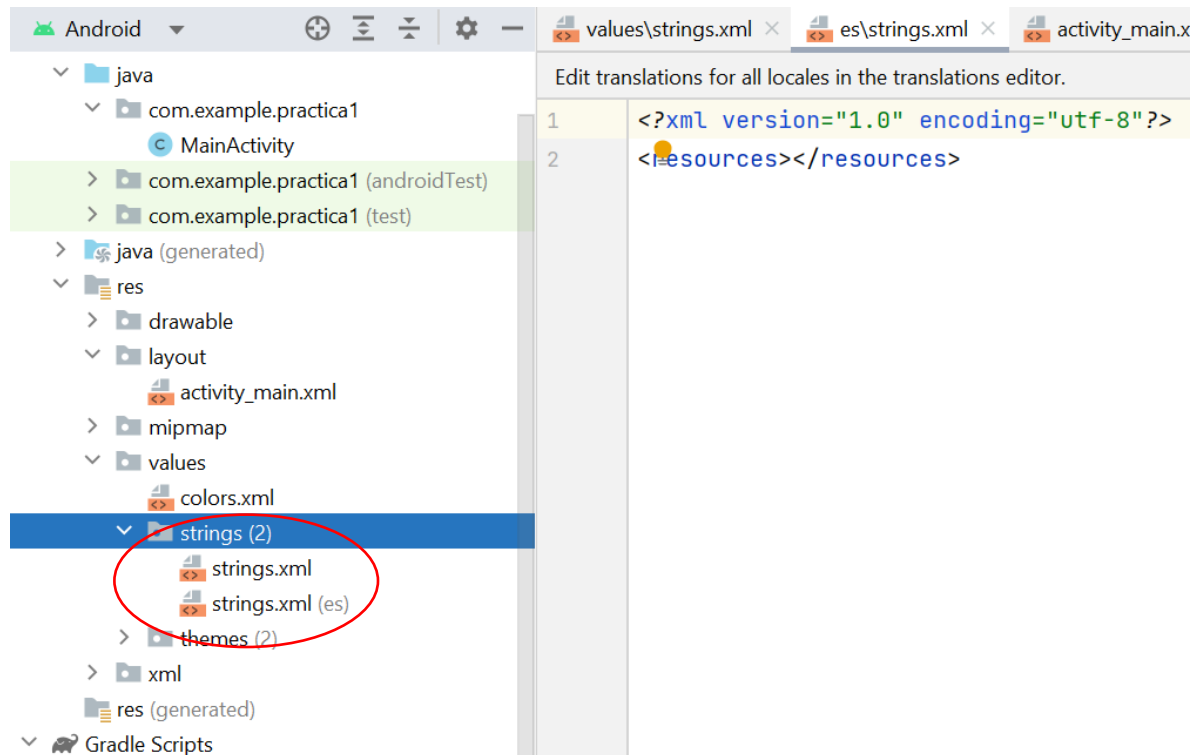


En el formulario emergente, nombraremos nuestro archivo como “strings.xml” y en el campo “Directory name:” escribimos el nombre de la carpeta values-es

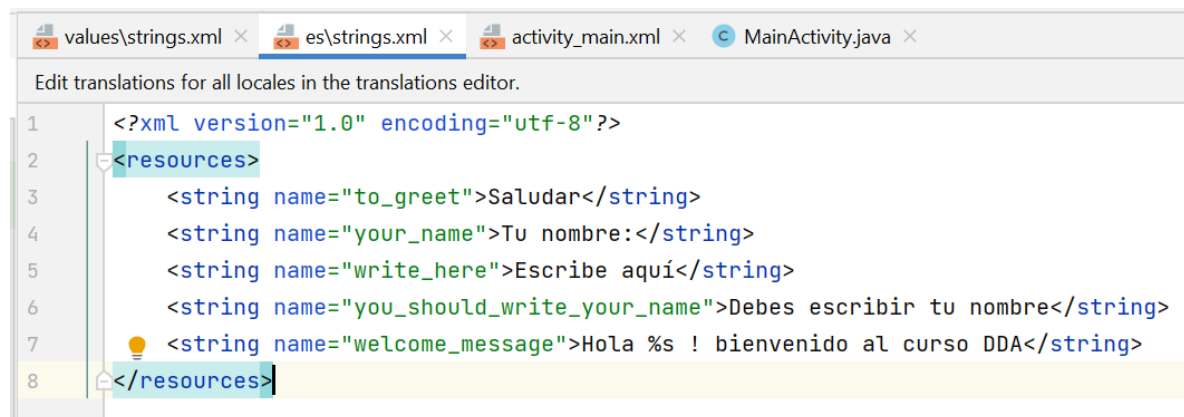


¹ https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

Al pulsar el botón OK, notaremos que ahora tenemos dos archivos string.xml en nuestro proyecto. El IDE los diferenciará por el sufijo de la carpeta.



Ahora copiaremos todos los recursos strings que ya habíamos generado y los pondremos en nuestro nuevo archivo xml y lo traduciremos. Nos quedará de la siguiente forma:



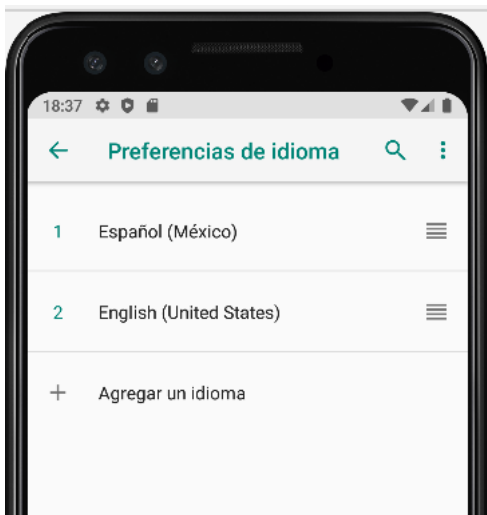
A continuación, notarás que en el archivo strings.xml original, aparece un error en el primer string "app_name". Esto se debe a que todos los strings deben ser traducidos. Pero tendremos casos en los que no es necesaria la traducción, como es el caso del nombre de la aplicación, la cual debería ser la misma en todos los idiomas.

```
values\strings.xml x es\strings.xml x activity_main.xml x MainActivity.java x
Edit translations for all locales in the translations editor.
1 <resources>
2   <string name="app_name">Practica1</string>
3   <string name="to_greet">To Greet</string>
4   <string name="your_name">Your name:</string>
5   <string name="write_here">Write here</string>
6   <string name="you_should_write_your_name">You should write your name</string>
7   <string name="welcome_message">Hi %s ! welcome to DDA course</string>
8 </resources>
```

Para solucionar el problema utilizaremos la propiedad `translatable="false"`, la cual le indica a Android que ese string no es necesario que tenga su versión traducida.

```
values\strings.xml x es\strings.xml x activity_main.xml x MainActivity.java x
Edit translations for all locales in the translations editor.
1 <resources>
2   <string name="app_name" translatable="false">Practica1</string>
3   <string name="to_greet">To Greet</string>
4   <string name="your_name">Your name:</string>
5   <string name="write_here">Write here</string>
6   <string name="you_should_write_your_name">You should write your name</string>
7   <string name="welcome_message">Hi %s ! welcome to DDA course</string>
8 </resources>
```

Para ver el resultado. Prueba con el emulador cambiando de idioma



Hecho esto regresa a la aplicación en el emulador y deberá verse en español

