

UD3.3 – Introducción a Android

2º CFGS
Desarrollo de Aplicaciones Multiplataforma
2023-24

1.- Centralizar valores en el proyecto

En la aplicación se han utilizado literales para los Strings y para los tamaños.

```
Text(  
    text = ";PÚLSAME!",  
    fontSize = 30.sp,  
    modifier = Modifier.padding(16.dp)  
)
```

A esta práctica se le denomina como **Hardcode values**.

Esto **no es una buena práctica**:

- ¿Qué pasa si se decide que todos los botones tengan el mismo tamaño de texto y en un futuro se cambia este tamaño?
- ¿Qué se debe hacer si se quiere crear una aplicación multi idioma?

1.- Centralizar valores en el proyecto

En Android Studio se pueden **centralizar valores** (similar a crear variables) para poder utilizar esos valores en cualquier punto del proyecto.

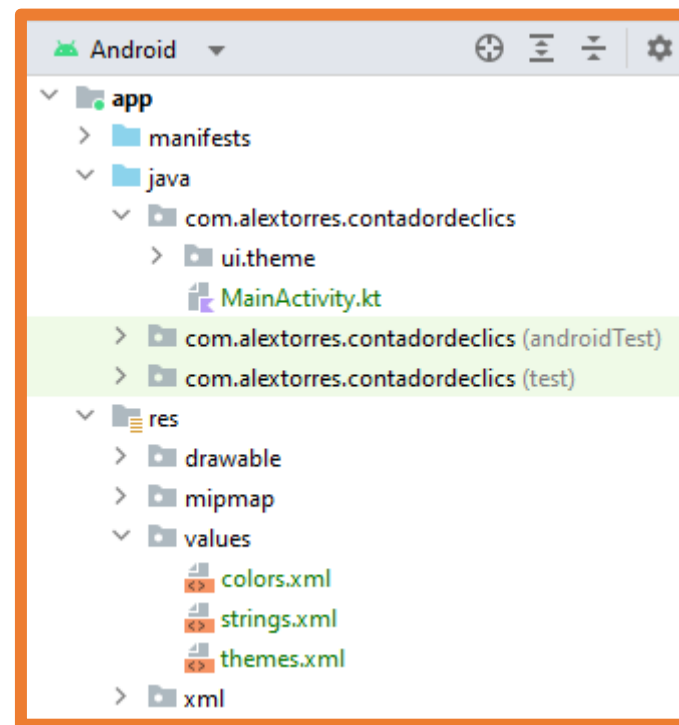
De esta manera los cambios serán más sencillos de realizar.

Como buena práctica se deberán centralizar al menos todas las cadenas de texto.

1.- Centralizar valores en el proyecto

Por defecto Android Studio ya crea dos archivos para estas tareas:

- colors.xml → centralizar **colores**
- string.xml → centralizar **cadena**s



A continuación, se verá cómo usarlo y como crear un archivo similar para las **medidas y tamaños**.

1.- Centralizar valores en el proyecto


Se pueden **crear colores y cadenas** añadiendo el código directamente a los **archivos correspondientes**, siguiendo el patrón ya establecido en ellos.

`<color name="NOMBRE">#CODIGO_COLOR</color>`

El código de color se puede representar mediante dos Dígitos hexadecimales para cada componente:

Sin transparencia: RedGreenBlue

Con transparencia: AlphaRedGreenBlue

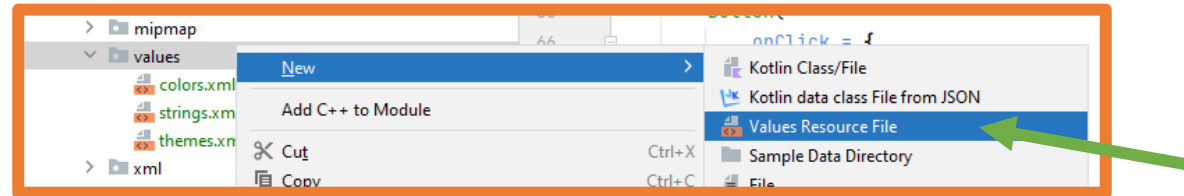
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
     <color name="white">#FFFFFFFF</color>
</resources>
```

`<string name="NOMBRE">CADENA</string>`

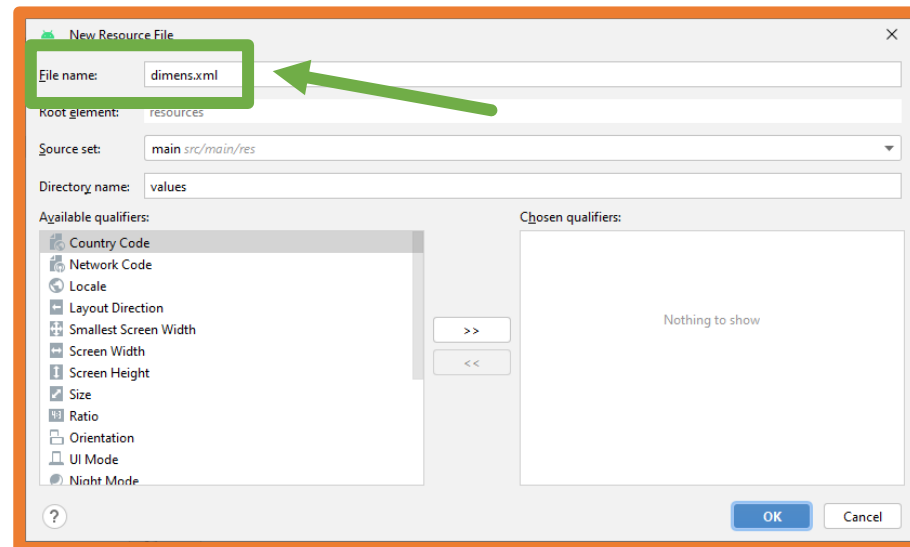
```
<resources>
     <string name="app_name">Contador de clics</string>
</resources>
```

1.- Centralizar valores en el proyecto

Para crear el archivo de dimensiones se deben seguir los siguientes pasos:
Clic derecho sobre **values** → **New** → **Values Resource File**.

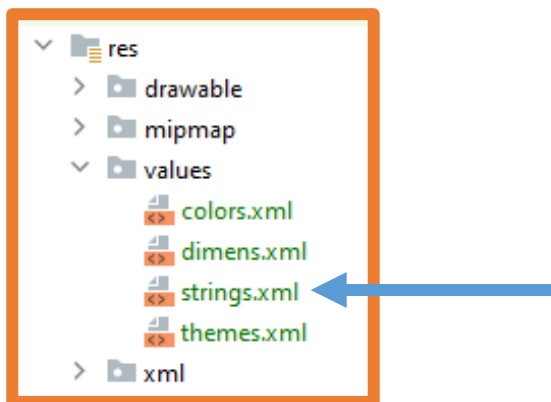


En la ventana que se abre se debe indicar el nombre del archivo: **dimens.xml** y el resto de opciones dejarlas por defecto.



1.- Centralizar valores en el proyecto

De esta manera se tendrá también el archivo **dimens.xml** disponible.



La estructura del archivo es similar al de los anteriores como se muestra en la siguiente imagen.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="button_text">25sp</dimen>
</resources>
```

Se pueden añadir todas las dimensiones que se necesiten.

2.- Uso de valores centralizados

Kotlin crea por defecto la clase **R** y la rellena con todos los recursos del proyecto (datos obtenidos de la carpeta **res**).

Así, una vez centralizados los valores se puede usando la clase **R** a todo lo que se ha definido en los archivos colors.xml, dims.xml, strings.xml.

```
R.string.counter_text
```

```
R.color.orange
```

```
R.dimen.button_text
```


2.- Uso de valores centralizados

Teniendo el acceso a los valores centralizados con la clase **R** se dispone de tres funciones para acceder a los valores centralizados según el archivo donde se encuentren:

- **stringResource**
- **dimensionResource** (si son sp se debe añadir detrás `.value.sp`)
- **colorResource**

```
Text(  
    text = "¡PÚLSAME!",  
    fontSize = 30.sp,  
    modifier = Modifier.padding(16.dp),  
)
```

```
Text(  
    text = stringResource(id = R.string.clickme),  
    fontSize = dimensionResource(id = R.dimen.button_text).value.sp,  
    modifier = Modifier.padding(dimensionResource(id = R.dimen.button_padding)),  
    color = colorResource(id = R.color.teal_200)  
)
```

3.- Añadir valores centralizados

Primera manera para añadir valores centralizados:

- Añadir directamente el valor al archivo XML correspondiente.

```
<resources>
  <string name="app_name">Contador de clics</string>
  <string name="greeting">Welcome to the application</string>
  <string name="clickme">¡PÚLSAME!</string>
</resources>
```

```
<resources>
  <dimen name="button_text">30sp</dimen>
  <dimen name="button_padding">16dp</dimen>
</resources>
```

```
<resources>
  <color name="purple_200">#FFBB86FC</color>
  <color name="purple_500">#FF6200EE</color>
  <color name="purple_700">#FF3700B3</color>
  <color name="teal_200">#FF03DAC5</color>
  <color name="teal_700">#FF018786</color>
  <color name="black">#FF000000</color>
  <color name="white">#FFFFFFFF</color>
  <color name="lightred">#FFFF2929</color>
  <color name="darkred">#FFAF0000</color>
</resources>
```

3.- Añadir valores centralizados

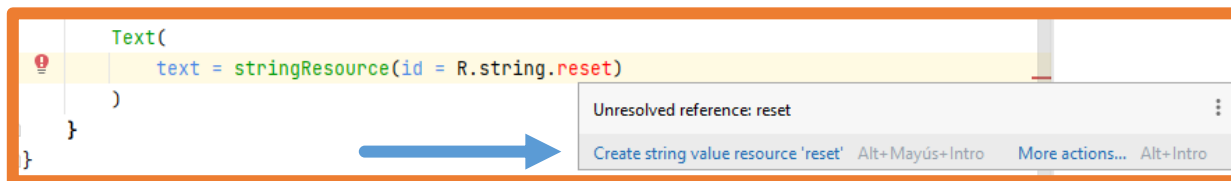
Segunda manera para añadir valores centralizados:

- En el código en Kotlin se puede indicar con una de las funciones anteriores un nuevo valor centralizado y desde ahí indicarle a Android Studio que añada el valor al XML.

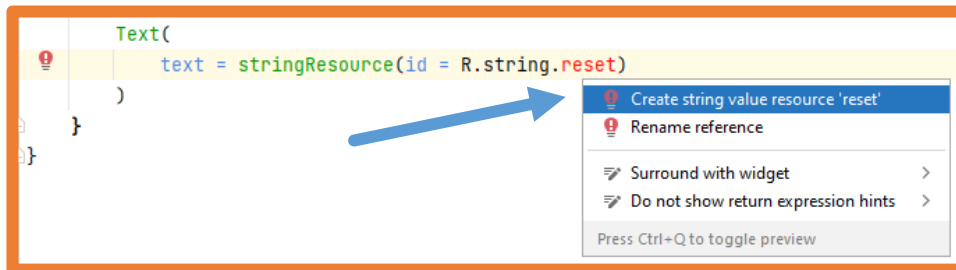
Al añadir el nuevo valor Android Studio lo marca como error:

```
Text(  
    text = stringResource(id = R.string.reset)  
)
```

Al situar el cursor encima del error aparece la ayuda contextual:



La ayuda contextual también aparece con el cursor encima del error al pulsar ALT+INTRO:

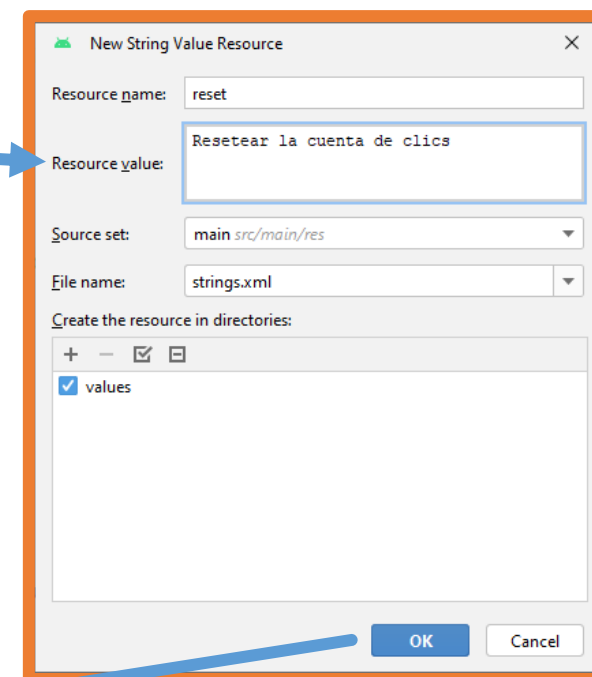


3.- Añadir valores centralizados

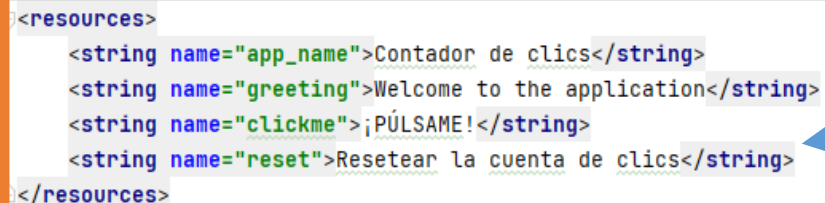
Segunda manera para añadir valores centralizados:

- En el código en Kotlin se puede indicar con una de las funciones anteriores un nuevo valor centralizado y desde ahí indicarle a Android Studio que añada el valor al XML.

Al pulsar **Create string value resource 'reset'** aparece la siguiente ventana en la que se debe rellenar el valor a centralizar.



Una vez añadido el valor y pulsado el botón **OK**, el valor se abrirá centralizado en su archivo.



```
<resources>
    <string name="app_name">Contador de clics</string>
    <string name="greeting">Welcome to the application</string>
    <string name="clickme">¡PÚLSAME!</string>
    <string name="reset">Resetea la cuenta de clics</string>
</resources>
```

4.- Aplicación multi idioma

Crear **aplicaciones multi idioma** con Android Studio es **muy sencillo**, simplemente se debe crear un archivo **strings.xml** para cada idioma soportado.

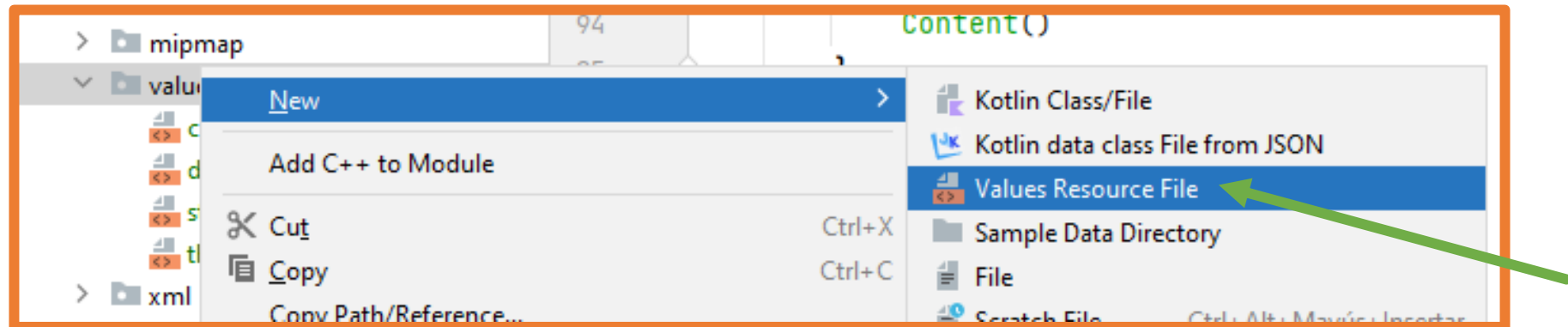
- Si solo se tiene un archivo **strings.xml** como ocurre nada más crear el proyecto, ese archivo se cargará sea cual sea el idioma configurado en el dispositivo.
- Si el dispositivo tiene un idioma configurado y no existe archivo **strings.xml** para ese idioma se cargará el archivo strings.xml por defecto (el que se crea junto al proyecto).
- Si el dispositivo tiene uno o más idiomas configurados y hay archivos **strings.xml** para esos idiomas, se cargará el primero de la lista de la configuración del móvil.
- Si se tienen varios archivos **strings.xml** y en uno no aparece una de las cadenas traducidas, se cargará la cadena en el idioma por defecto

Lo más habitual es que el archivo strings.xml creado con el proyecto sea para el idioma inglés.

4.- Aplicación multi idioma

Para crear un archivo strings.xml para un idioma se deben seguir los siguientes pasos:

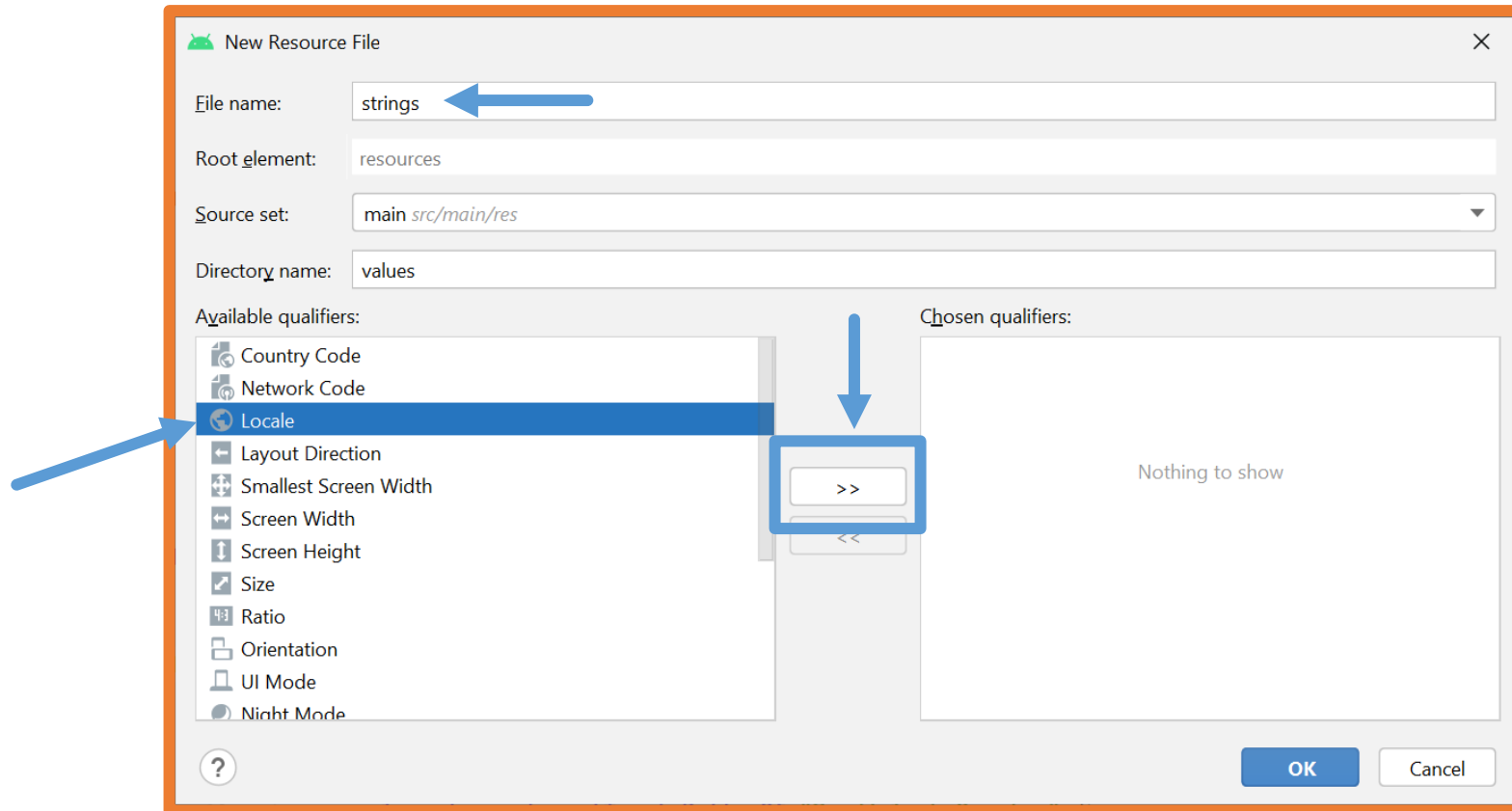
Clic derecho sobre la carpeta **values** → **New** → **Values Resource File**



4.- Aplicación multi idioma

Nombre de archivo: **strings**

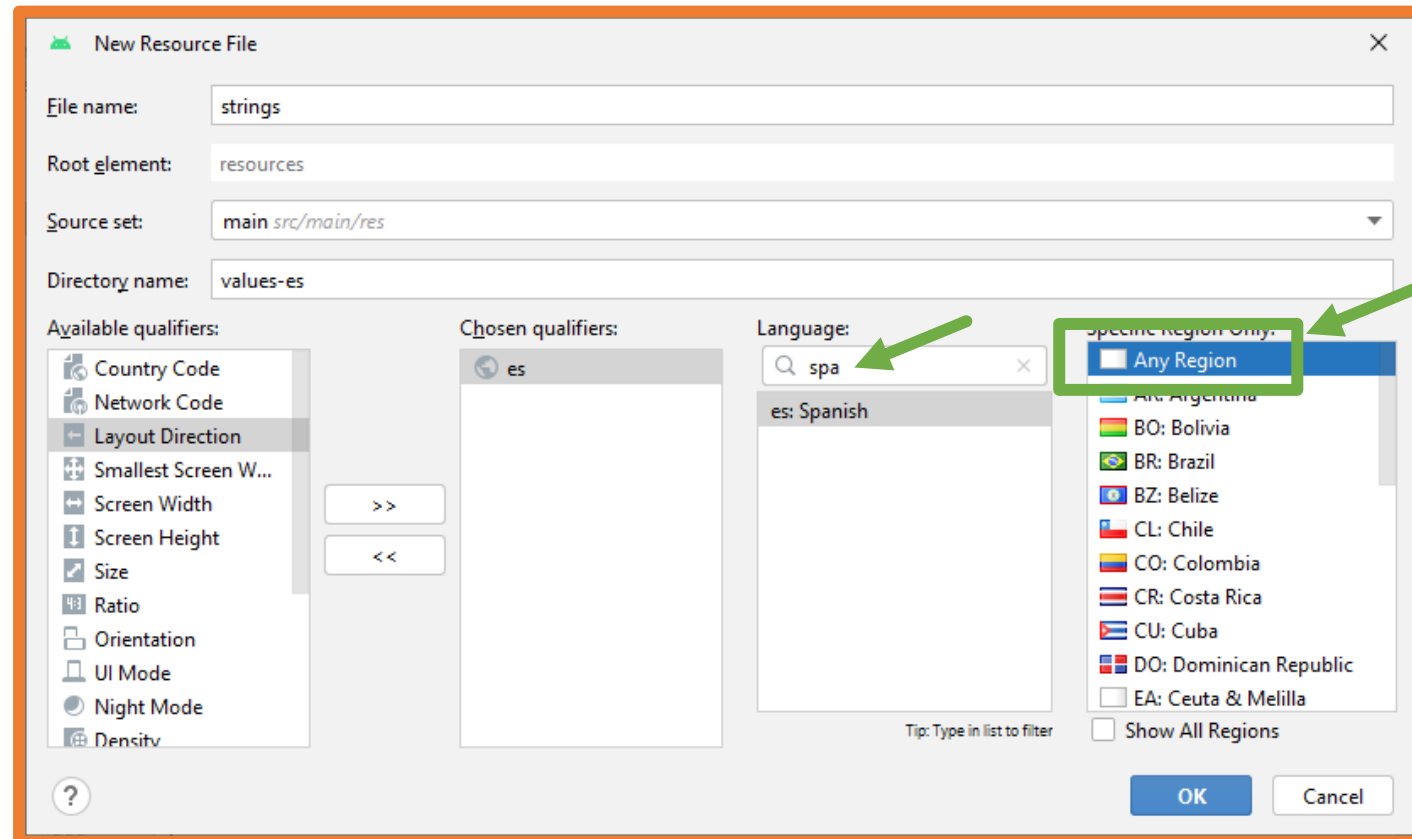
Seleccionar **Locale** y pulsar el botón >>



4.- Aplicación multi idioma

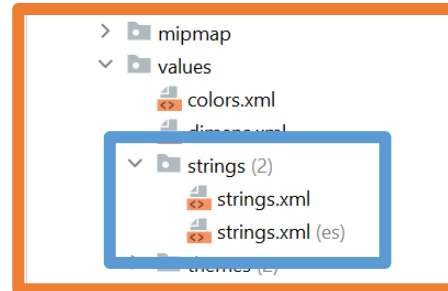
Seleccionar el idioma y región deseada.

Pulsar el botón **OK**

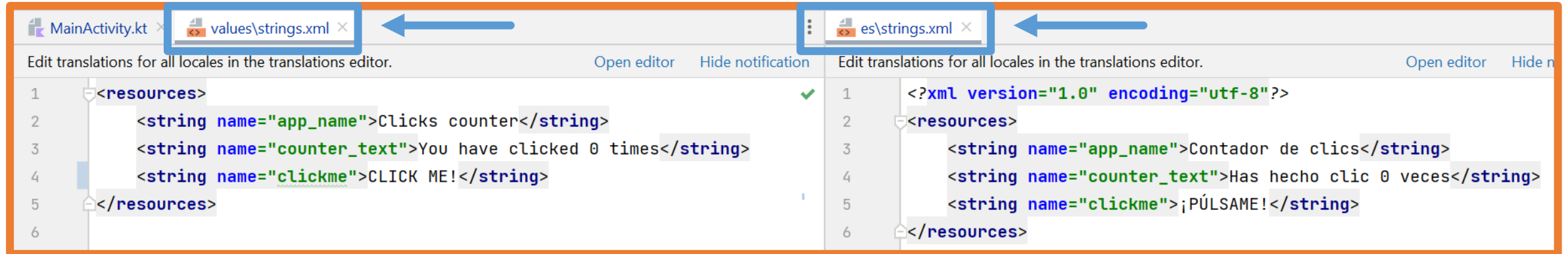


4.- Aplicación multi idioma

En el inspector del proyecto se puede ver cómo se ha creado un segundo archivo para centralizar cadenas:

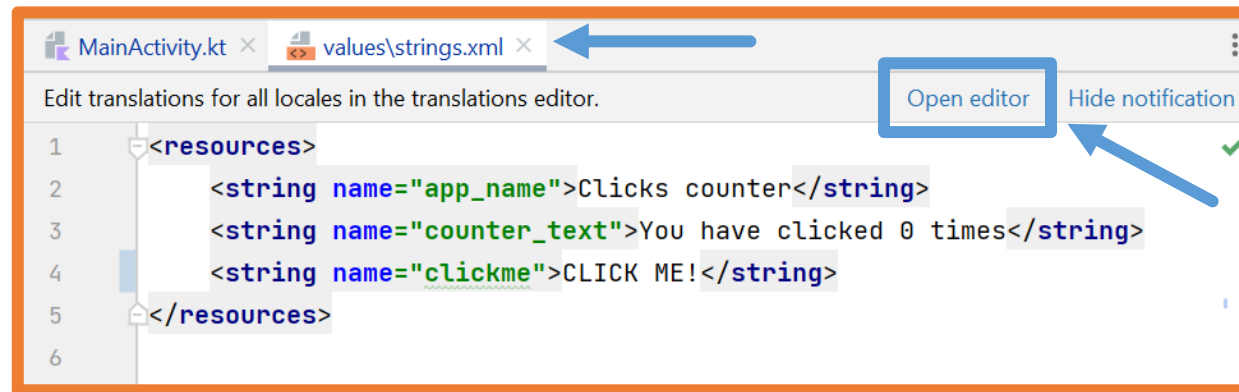


El siguiente paso es replicar el contenido del primer archivo strings.xml al segundo y traducir el valor de las cadenas:

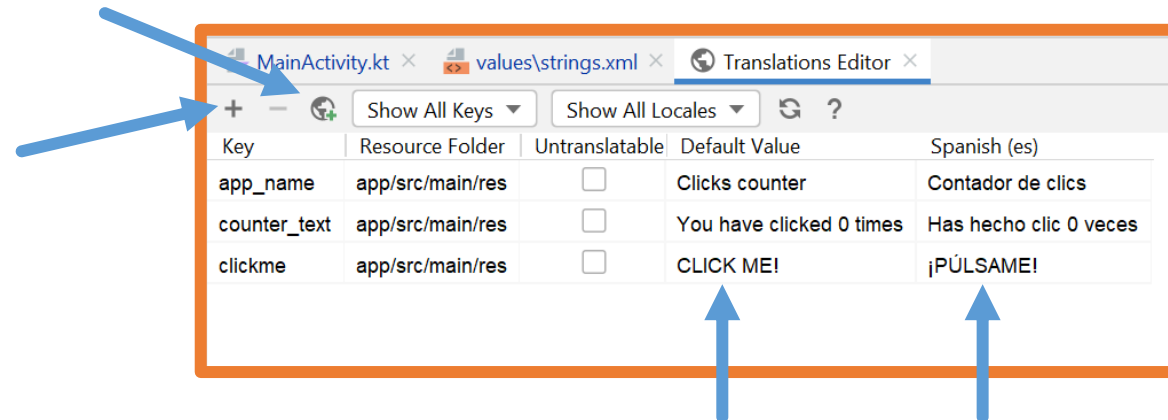


4.- Aplicación multi idioma

Mediante el editor de idiomas también se pueden añadir cadenas para los diferentes idiomas e incluso añadir nuevos archivos **strings.xml**:

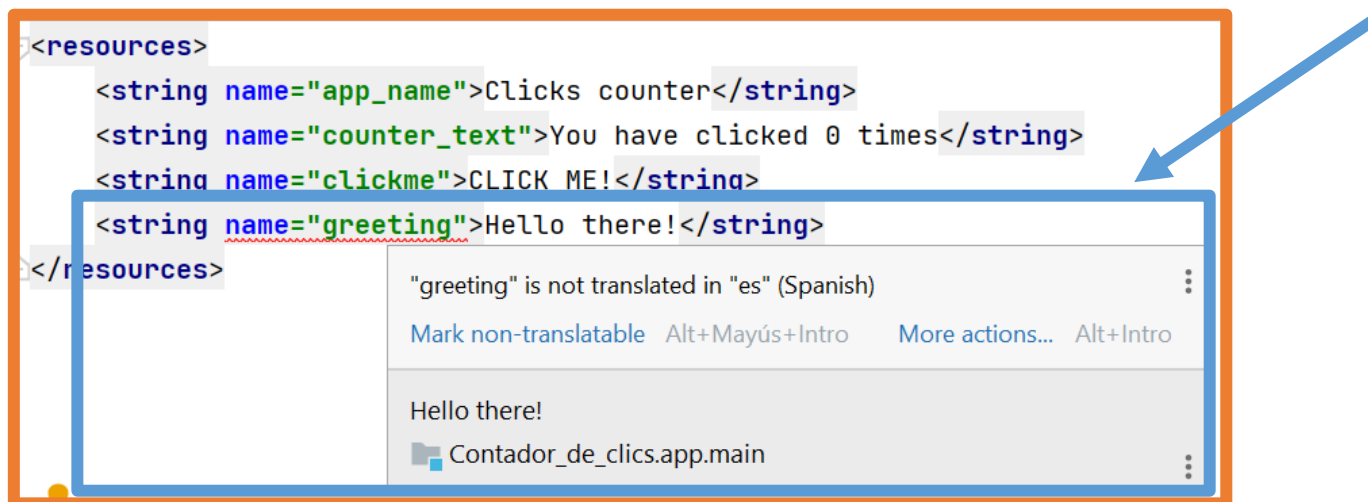


En el editor aparecen tantas columnas como archivos strings.xml haya:



4.- Aplicación multi idioma

En el momento en el que se tienen diferentes archivos strings.xml para diferentes idiomas, al añadir una cadena a uno de los archivos Android Studio avisará que se debe traducir a los otros idiomas.



Si una cadena no se traduce a algún idioma Android Studio mostrará la cadena definida en el archivo strings.xml por defecto.

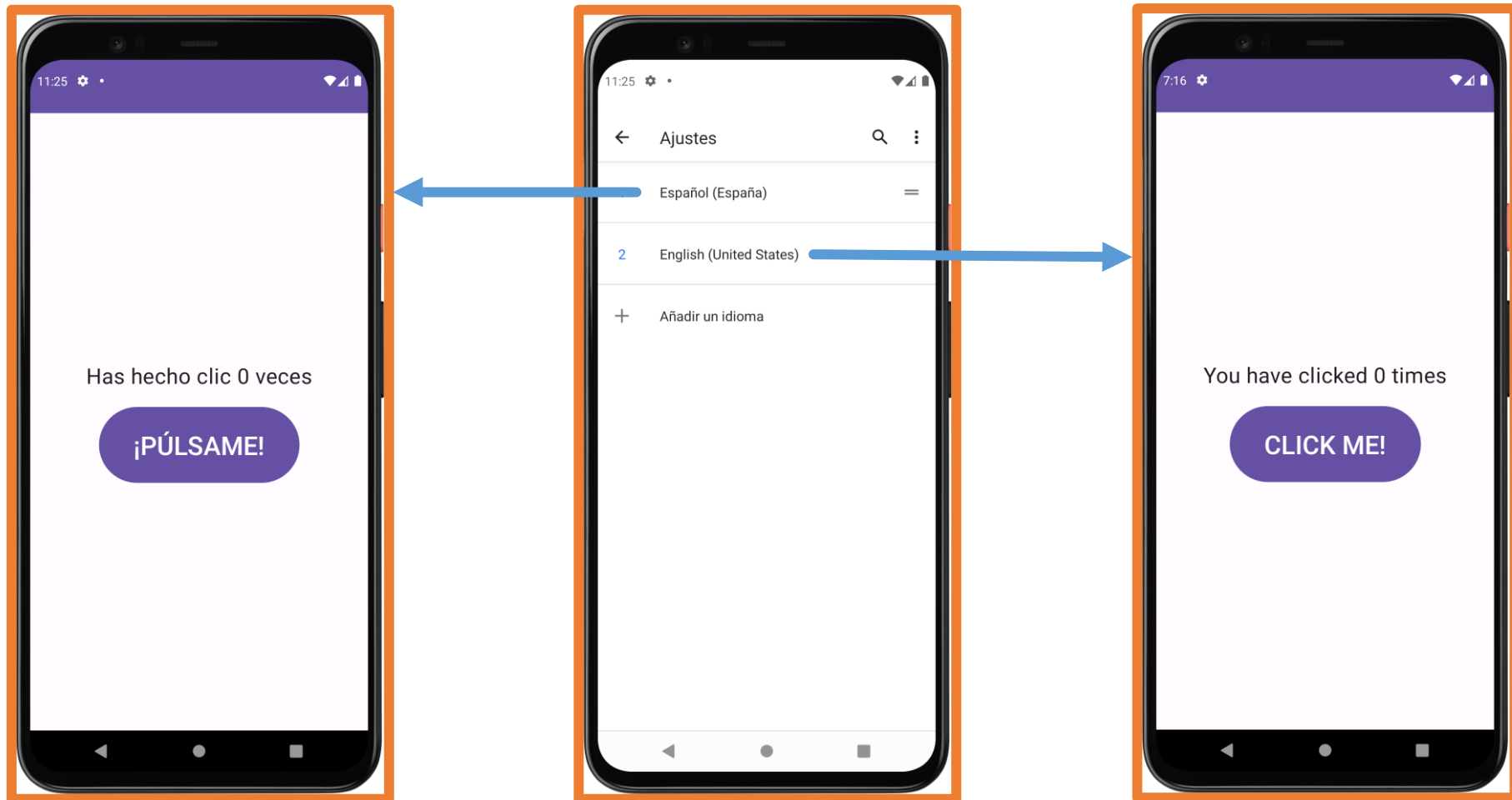
4.- Aplicación multi idioma

El resultado se puede ver con las previsualizaciones @Preview:



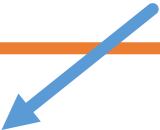
4.- Aplicación multi idioma

Y también se puede comprobar con el emulador:



5.- Valores centralizados que aceptan parámetros

En la aplicación **Contador de clics** se tiene la necesidad de centralizar la siguiente cadena:



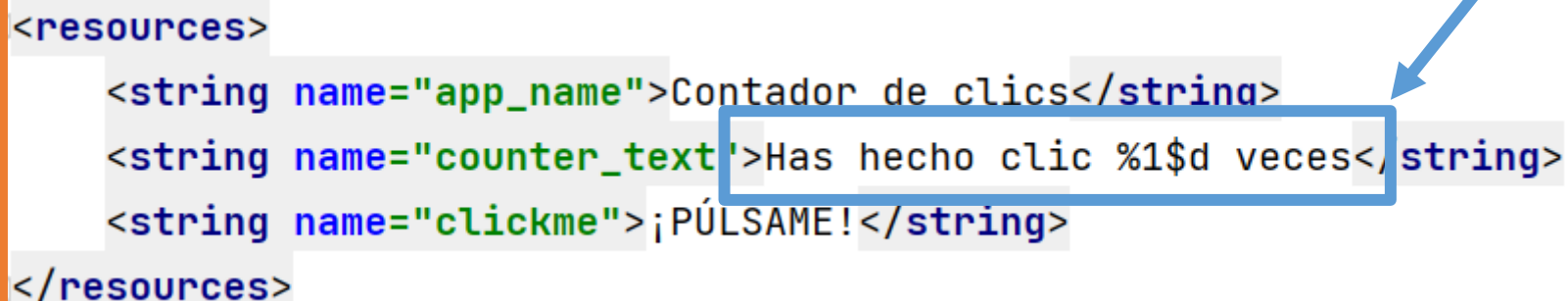
```
Text(  
    text = "Has hecho clic $times veces",  
    fontSize = 25.sp  
)
```

Esta cadena utiliza una variable por lo que a la hora de centralizar dicha cadena se debe pasar ese valor como parámetro.

5.- Valores centralizados que aceptan parámetros

El formato para declarar parámetros es **%NºParámetro\$Tipo**, siendo los tipos de parámetros: s (cadenas), d (números).

```
<resources>
    <string name="app_name">Contador de clics</string>
    <string name="counter_text">Has hecho clic %1$d veces</string>
    <string name="clickme">¡PÚLSAME!</string>
</resources>
```



Por ejemplo, si se quieren recibir varios parámetros:

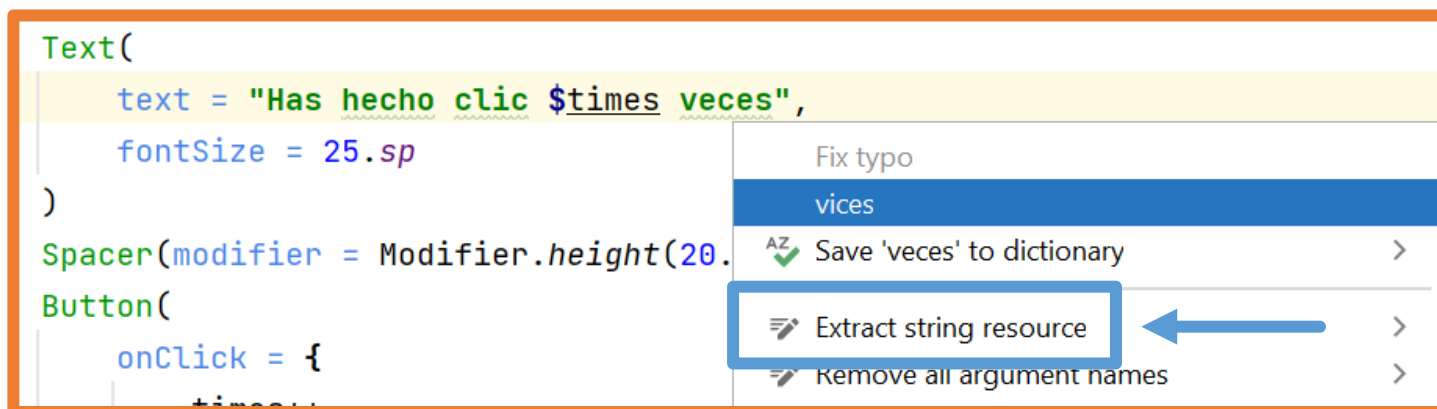
```
<string name="user_points_text">Hola %1$s, has conseguido %2$d</string>
```

En las últimas versiones de Android Studio se puede indicar siempre que el tipo de dato es s (cadena) ya que Android Studio se encarga de convertir automáticamente el dato a String.

5.- Valores centralizados que aceptan parámetros

Android Studio ayuda a crear estos valores centralizados.

Con el cursor encima de la **string hardcoded** se muestra la ayuda contextual con clic derecho o con ALT+INTRO y se selecciona la opción **Extract string resource**:

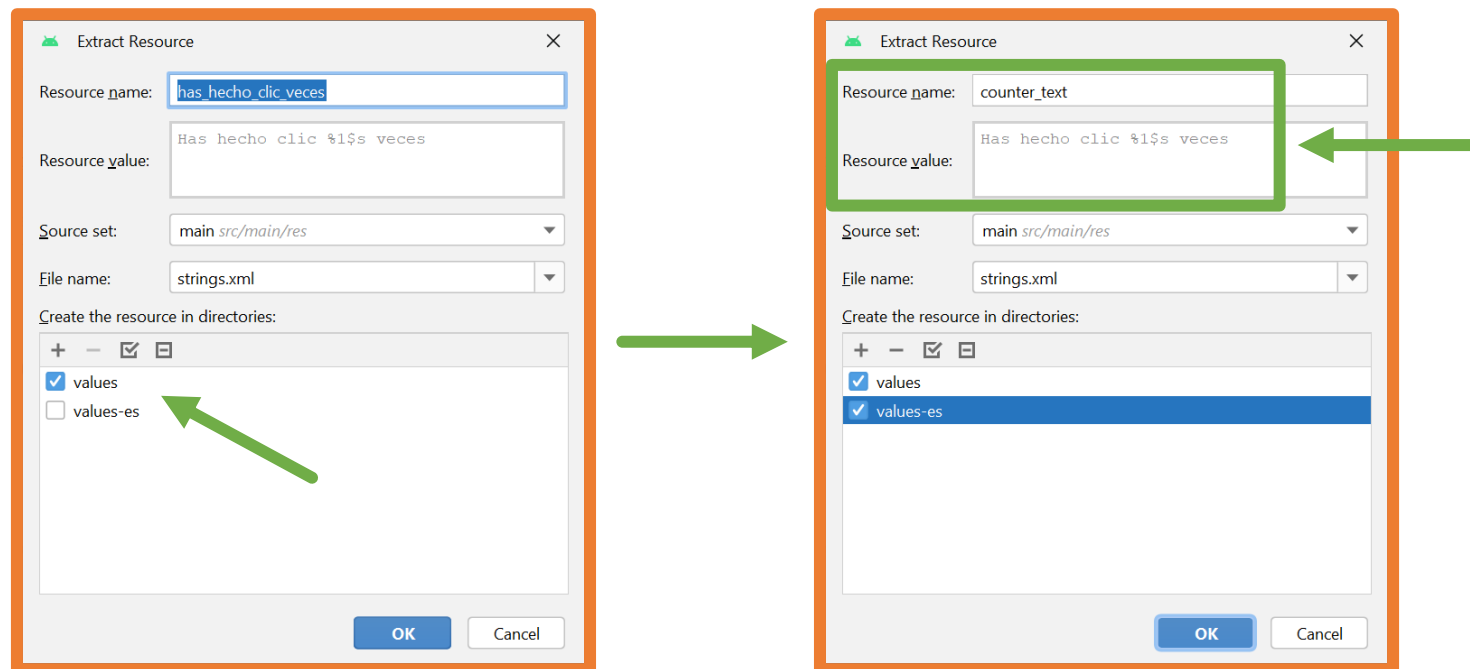


5.- Valores centralizados que aceptan parámetros

Android Studio ayuda a crear estos valores centralizados.

Con el cursor encima de la cadena hardcodeada se muestra la ayuda contextual con clic derecho o con ALT+INTRO y se selecciona la opción **Extract string resource**.

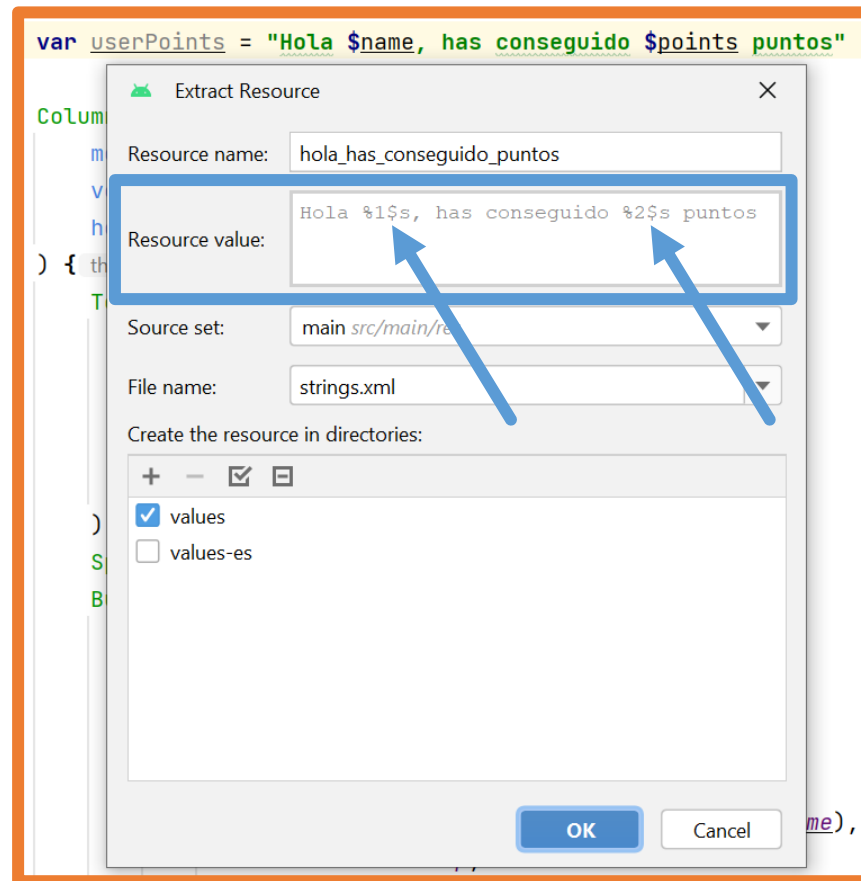
Si se tienen varios archivos strings.xml se puede marcar que se añada el valor a todos los archivos y una vez se añada solo habrá que traducir el valor en cada archivo.



5.- Valores centralizados que aceptan parámetros

Android Studio ayuda a crear estos valores centralizados.

Si la cadena tiene varios parámetros Android Studio los reconocerá:



5.- Valores centralizados que aceptan parámetros

Ahora desde Kotlin en la función **stringResource** se podrán pasar esos parámetros:

```
Text(  
    text = stringResource(  
        R.string.counter_text,  
        times  
    ),  
    fontSize = 25.sp  
)
```

```
var userPoints = stringResource(  
    R.string.user_points,  
    name,  
    points  
)
```

IMPORTANTE

La traducción de aplicaciones tiene un gran inconveniente ya que la misma frase no tiene la misma longitud en todos los idiomas.

Se debe realizar un buen estudio de la aplicación y de los diferentes idiomas para que la aplicación se muestre de manera similar en todos los idiomas en los que se distribuya.

6.- Orientaciones vertical y horizontal

En los dispositivos fijos como televisiones o automóviles no tiene sentido cambiar la orientación, de hecho directamente no disponen del acelerómetro para detectarla.

La mayoría de dispositivos móviles Android (móviles y tablets) permiten cambiar la orientación de la pantalla entre vertical y horizontal.

Existen excepciones como los smartWatch que siendo móviles generalmente solo tienen una orientación.

6.- Orientaciones vertical y horizontal

Si se desarrolla una aplicación para móvil o tablet **es importante tener en cuenta el diseño en las dos orientaciones.**

Gracias al uso de Jetpack Compose los componentes se van a situar ocupando la pantalla tal y como se definan.

Por esta razón, en principio todas las aplicaciones se visualizarán correctamente indistintamente del tamaño y la orientación de la pantalla.

Más adelante se verá cómo mostrar diferentes componentes o los mismos componentes con diferente estilo según el tamaño de pantalla para crear una mejor experiencia de usuario.

6.- Orientaciones vertical y horizontal

Es posible que se quiera diseñar una aplicación que no permita cambiar la orientación.

Para ello en el archivo **AndroidManifest.xml** se debe añadir la siguiente propiedad a la Activity:

`android:screenOrientation="ORIENTACIÓN"`

Esta configuración se debe hacer para todas las Activities de la aplicación en las que se quiera bloquear la orientación.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.ContadorDeClicks"
        tools:targetApi="31">

        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:screenOrientation="portrait"
            android:label="@string/app_name"
            android:theme="@style/Theme.ContadorDeClicks">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

7.- Values Resource Files

Hasta este punto se han utilizado los **archivos de recursos con valores** para centralizar valores (strings.xml, colors.xml y dims.xml).

Estos archivos también se pueden utilizar para que se carguen dependiendo de la configuración del dispositivo.

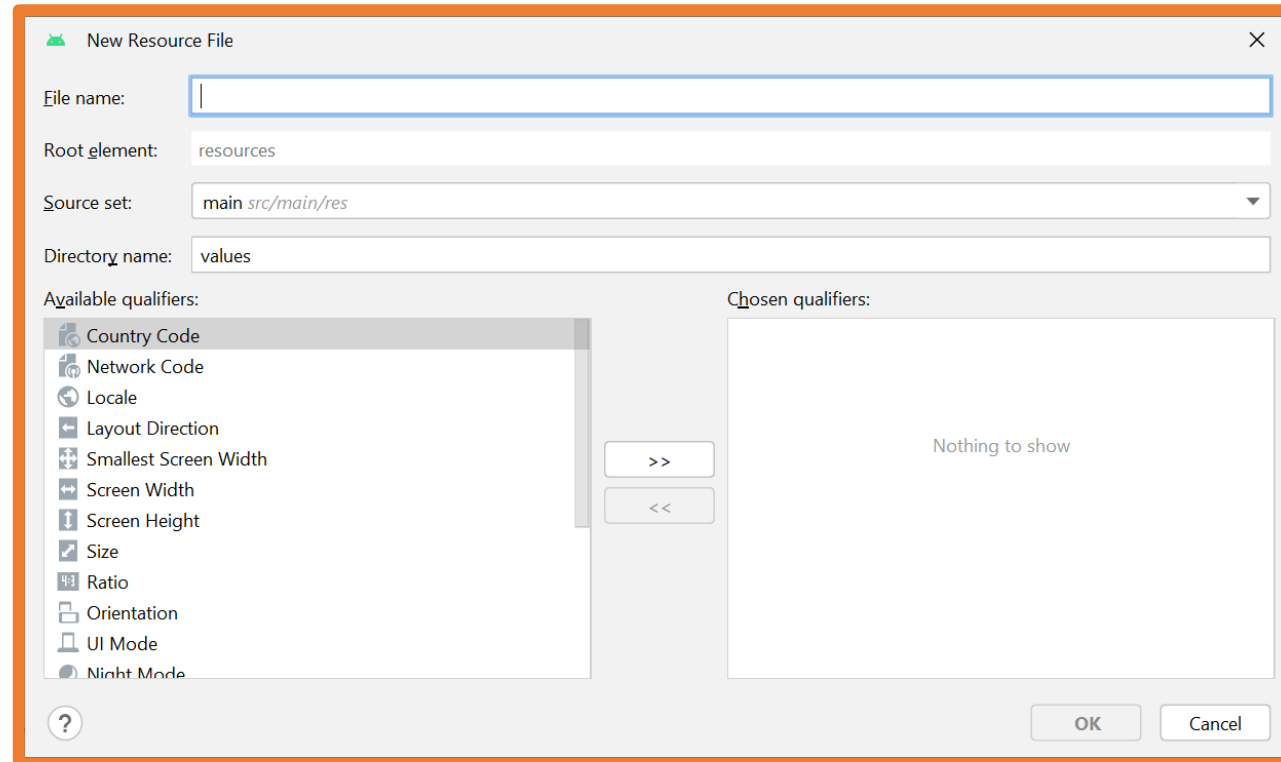
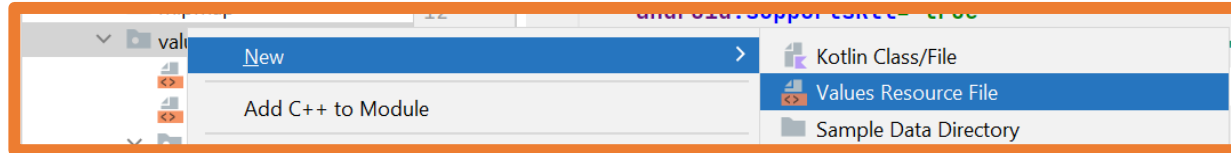
Por ejemplo, se puede cargar un archivo colors.xml cuando la orientación es vertical (portrait) y otro diferente cuando la orientación es horizontal (landscape). Igual que ocurre con los archivos strings.xml dependiendo del idioma.

Con la manera antigua de implementar las interfaces de usuario mediante archivos XML esta configuración tenía mucho más sentido.

A continuación, se va a estudiar cómo crear estos archivos.

7.- Values Resource Files

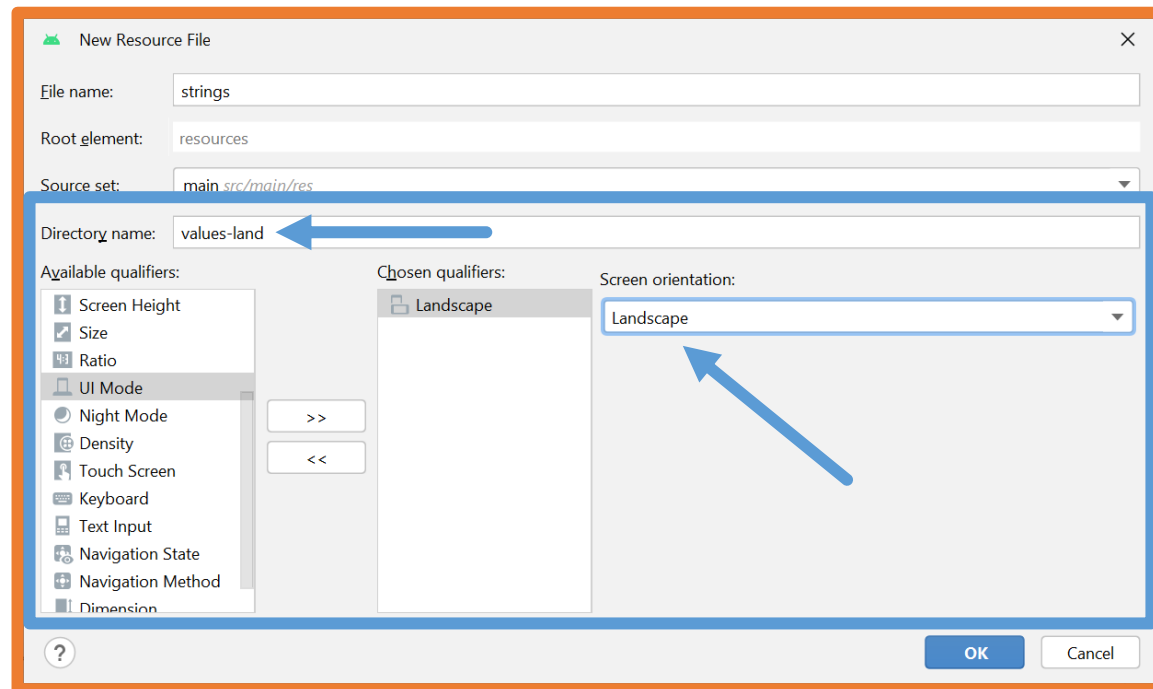
Sobre la carpeta **values** se hace clic con el botón derecho y se selecciona la opción **Values Resource File**.



7.- Values Resource Files

El funcionamiento de estos archivos es el mismo que el comportamiento de los archivos strings.xml.

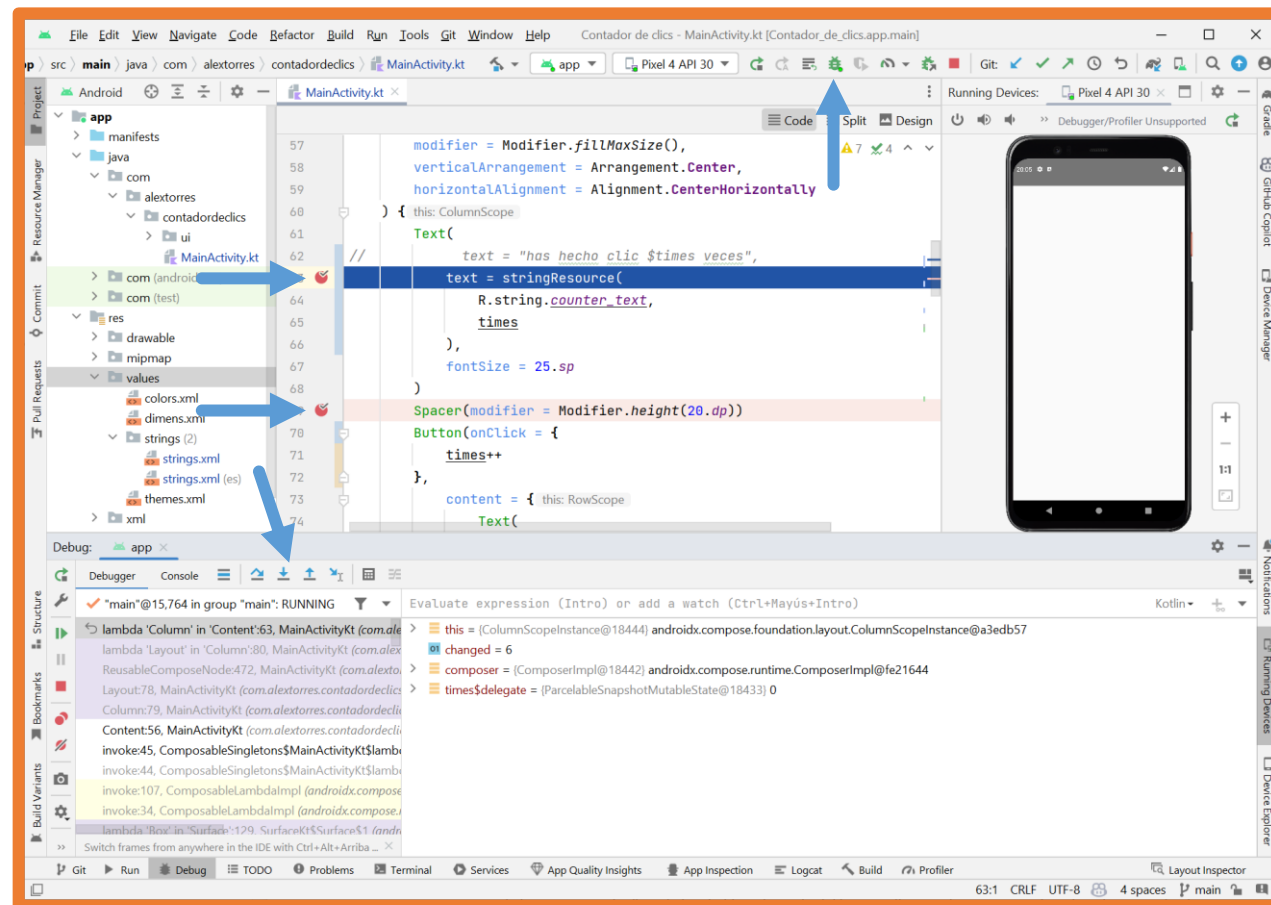
Se creará el archivo en una carpeta que Android cargará según la configuración del dispositivo.



8.- Debug

Android Studio dispone de un **modo depuración** que funciona como en el resto de IDE's.

Se pueden configurar **puntos de ruptura**, ejecutar en modo depuración, avanzar paso a paso...

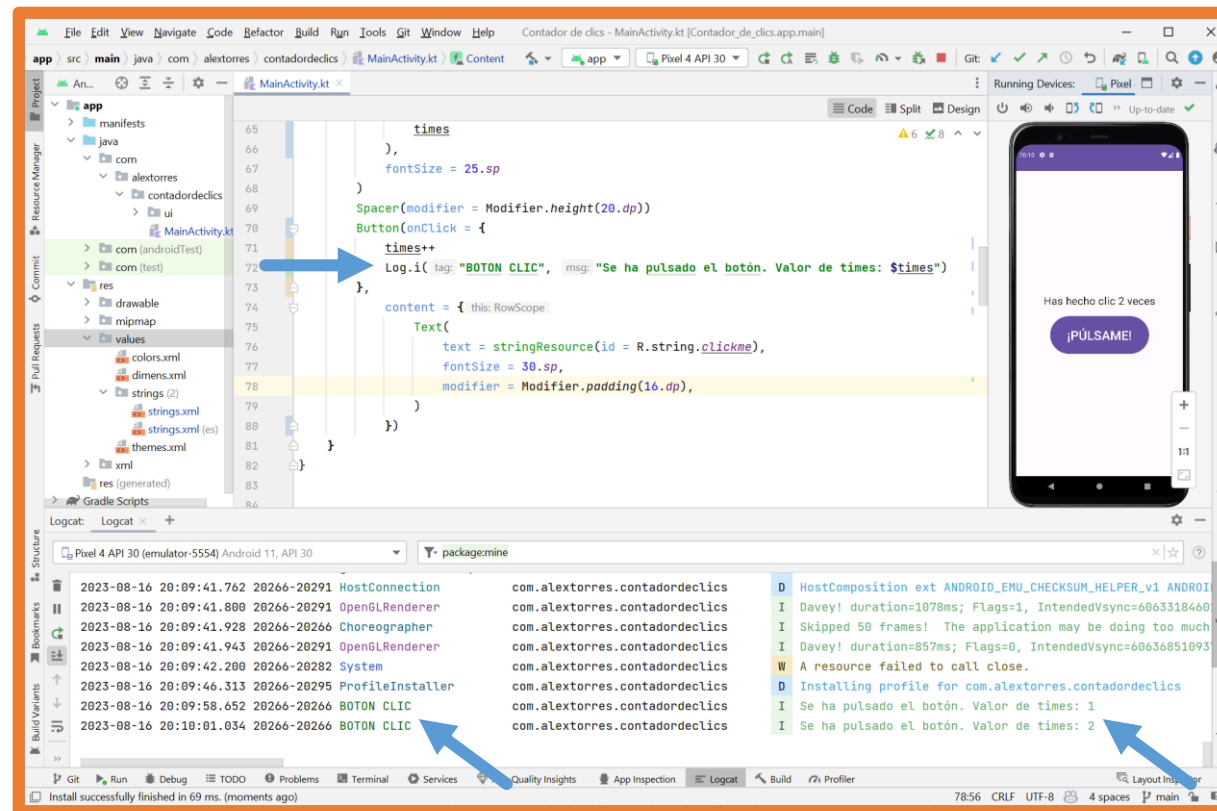


8.- Debug

Para comprobaciones más básicas se puede utilizar la clase **Log** de la siguiente manera:

```
Log.i("ETIQUETA", "MENSAJE")
```

De esta manera, el mensaje aparecerá en la sección **Logcat** de Android Studio.



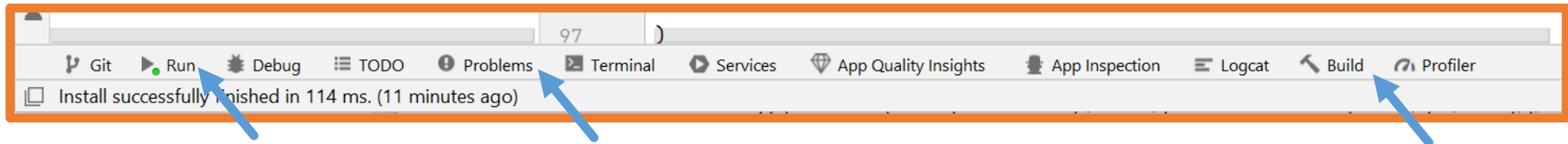
9.- Actualizaciones de Android Studio

Software **Android Studio** recibe actualizaciones que pueden ser de diferentes componentes:

- Android Studio.
- SDK de Android.
- JDK.
- Plugins.
- Gradle.
- Dependencias.

En ocasiones cuando se actualizan componentes hay proyectos que dejan de funcionar hasta que no se configure correctamente.

La manera de mostrar los error dependerá del tipo de actualización que lo ha provocado.



Según el tipo de error la solución se realizará de una manera u otra.

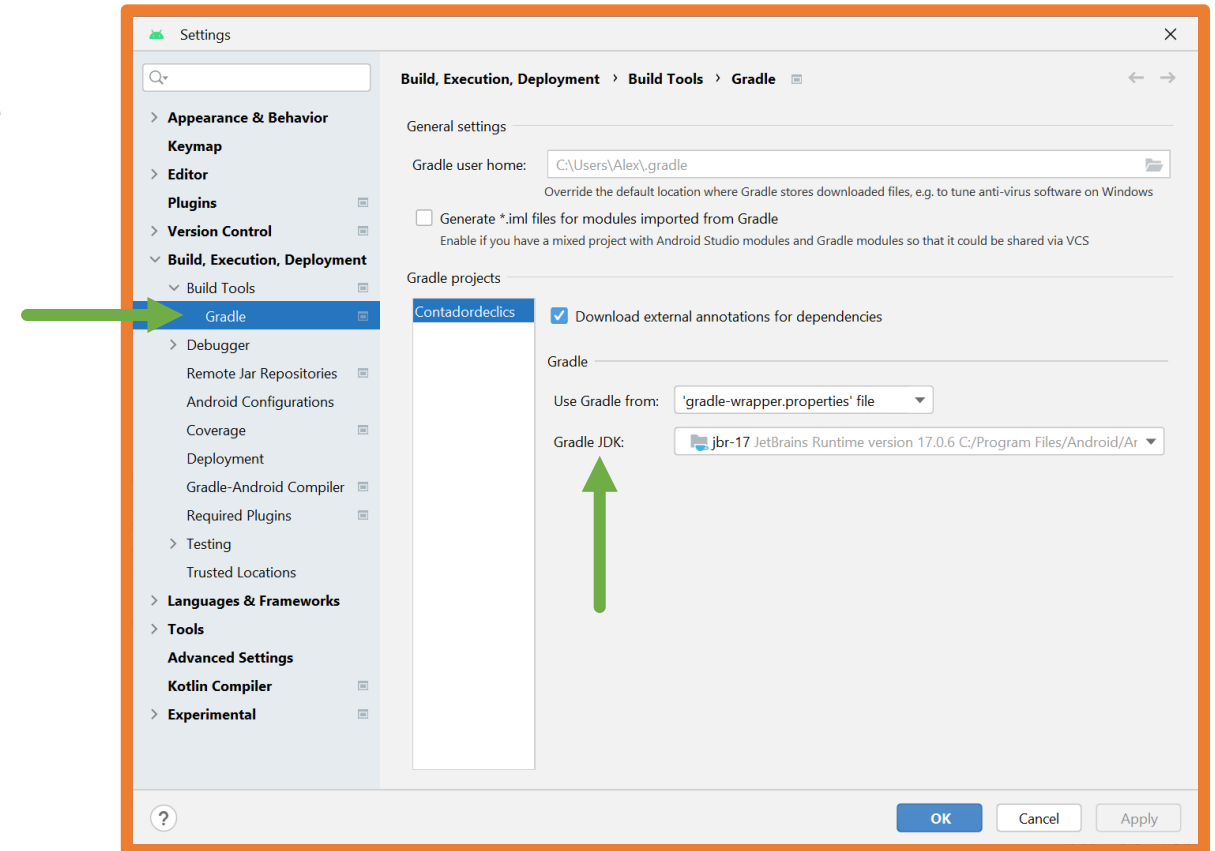
9.- Actualizaciones de Android Studio

Error versión de Java:

Para solucionar el error de versión de Java se debe ir a la configuración:

File → Settings (CTRL+ALT+S)

Y seleccionar el JDK correcto.



9.- Actualizaciones de Android Studio


Error **appcompat**:

En las aplicaciones diseñadas XML se utiliza la librería **appcompat**.

Si se está desarrollando una aplicación que mezcla XML y Jetpack Compose, por ejemplo si se están migrando de XML a Jetpack Compose también se utiliza esa librería.

En ocasiones al actualizar componentes se debe cambiar la versión de la dependencia, esto se configura en el archivo **build.gradle (Module: app)**:

```
dependencies {  
    implementation 'androidx.core:core-ktx:1.7.0'  
    implementation 'androidx.appcompat:appcompat:1.6.0'  
    implementation 'com.google.android.material:material:1.8.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
```



Práctica

Actividad 3

Contador de pádel.