

Views: Tipos de Views más habituales y sus propiedades.

Contenido

1	Introducción a vistas.	2
1.1	Layouts	2
1.2	Widgets.....	3
1.3	Date & Time	4
1.4	Expert.....	5
1.5	Custom	6
2	Propiedades de los Views	7
3	Vistas de Texto	9
3.1	TextView Android	10
3.2	EditText Android	13
3.2.1	Código Java para los TextView y EditText	14
4	Los Botones en Android.....	16
4.1	Ejemplo Button Android	17
4.2	Ejemplo ToggleButton Android	21
4.2.1	Practica. Intentar añadir un ImageButton como el siguiente	22
4.3	Listener para Button de Android en el código Java.....	24
4.4	Listener para Toggle Button.....	25
4.5	Actividad.	26
5	Spinner	26
5.1	Creando el Spinner	26
5.2	Introduciendo datos en el Spinner desde Java con un array.....	29
5.2.1	Actividad de Spinner.....	30
5.3	Seleccionando el cambio de opción de un spinner.....	31

1 Introducción a vistas.

Como vamos a ver, es posible colocar los **Views o elementos de interacción de una actividad** arrastrándolos desde el menú a la Activity correspondiente en Android Studio de manera que podamos modificar todas sus propiedades y colocarlo en función a lo que más nos convenga. Sin embargo, esto no deja de ser solo una **parte de las propiedades o atributos** que podemos configurar a un elemento o View.

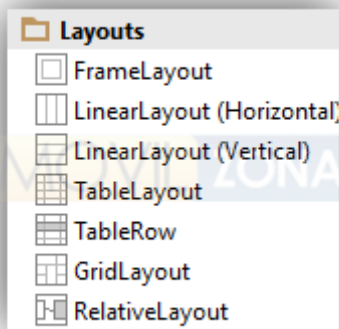
Las vistas o **Views** son todos aquellos **elementos de una aplicación** para Android que, **dentro** de una **Activity**, sirven para que el usuario interactúe con la aplicación. Dentro de los Views podemos distinguir **varios elementos**, siendo los más comunes las etiquetas, los **cuadros de texto**, las **casillas y los botones**, entre otros.

Android Studio clasifica los Views en varias categorías según su finalidad para que los usuarios puedan encontrar lo que buscan fácilmente:

1.1 Layouts

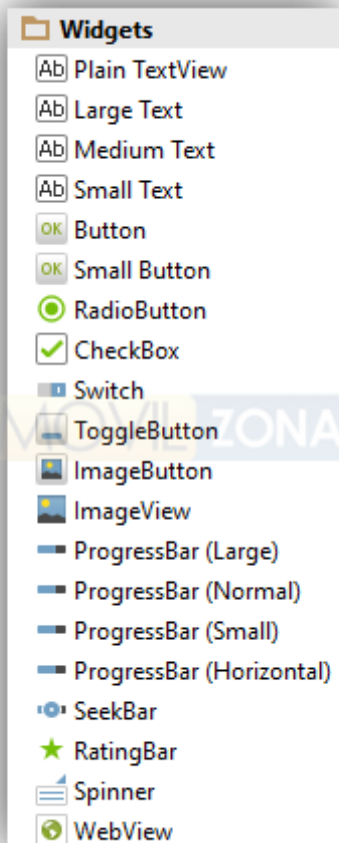
Un **Layout es el elemento encargado** de representar el diseño de la **interfaz** con la que va a interactuar el usuario. Dentro de ella tendremos los elementos gráficos, los Views, información sobre la actividad, fragmentos e incluso los widgets.

Dentro de este apartado podemos **cargar nuevos Layouts** que modifiquen el comportamiento de la actividad.



1.2 Widgets

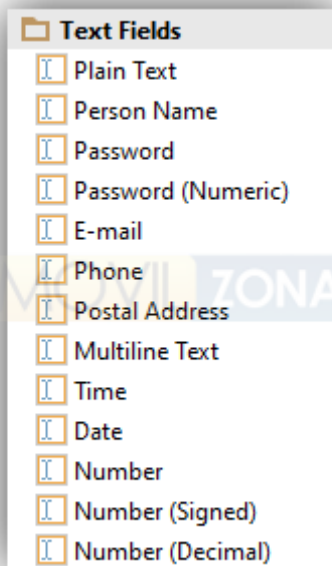
Dentro de este apartado encontraremos una serie de elementos diseñados para que el usuario pulse sobre ellos o para mostrar información al usuario sobre algo. Por ejemplo, dentro de **Widgets** encontraremos cuadros de texto, botones, casillas, interruptores, barras de progreso, etc.



TextFields

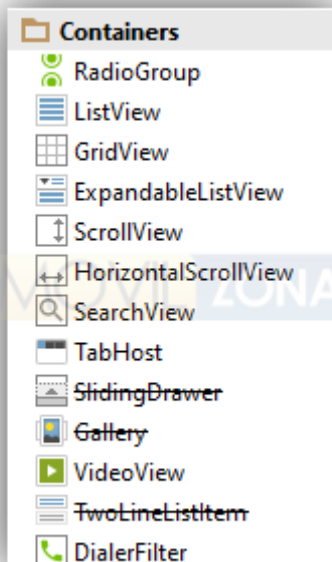
Los **TextFiles** son elementos diseñados para que el usuario introduzca en ellos un tipo de texto como un nombre, un teléfono, una respuesta a una pregunta o

cualquier otro tipo de información. Existen varios tipos de **TextFields**, siendo algunos de los más habituales PlainText, Person Name, Password, Email, Phone o Multiline Text.



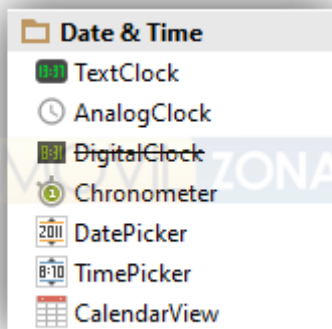
Containers

Dentro de este apartado vamos a encontrar los contenedores, es decir, grupos de elementos. Aquí podremos encontrar listas y grupos de Views **barras de búsqueda, filtros, etc.**



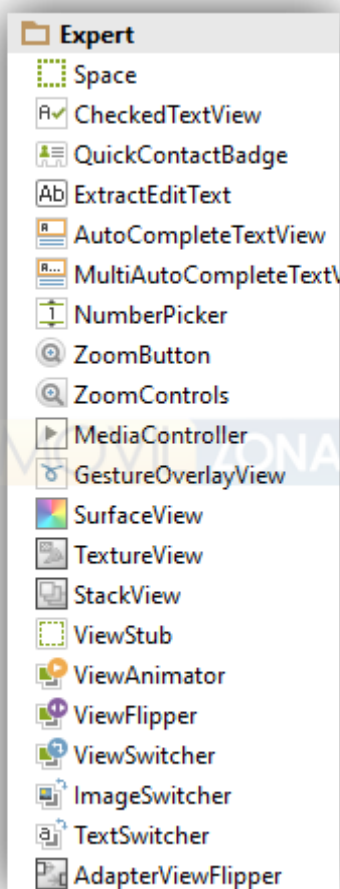
1.3 Date & Time

Dentro de este apartado, como su nombre indica, encontraremos todo lo relacionado con el tiempo, es decir, **relojes, cronómetros, calendarios**, etc.



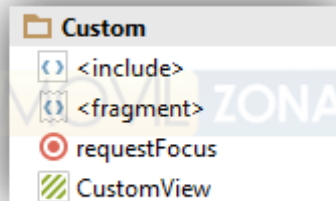
1.4 Expert

Aquí encontraremos una serie de Views orientados a usuarios expertos debido a su complejidad de uso.



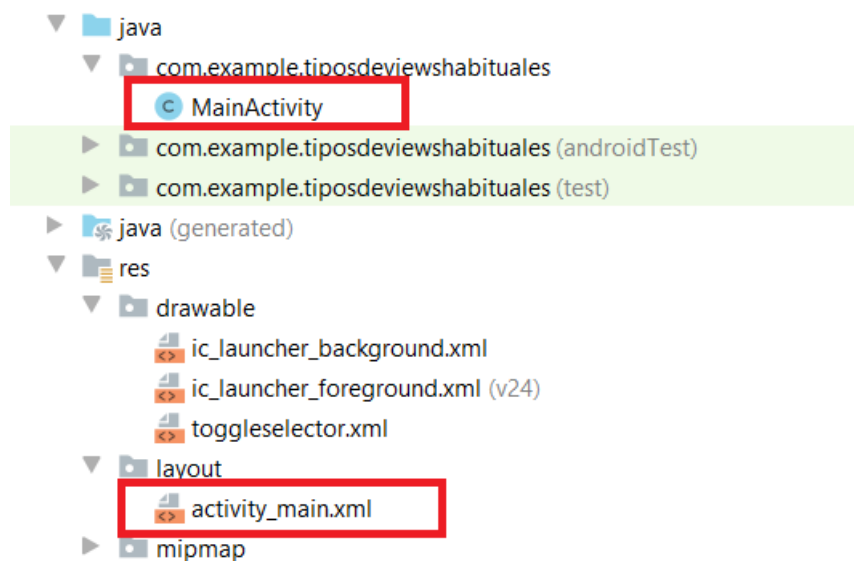
1.5 Custom

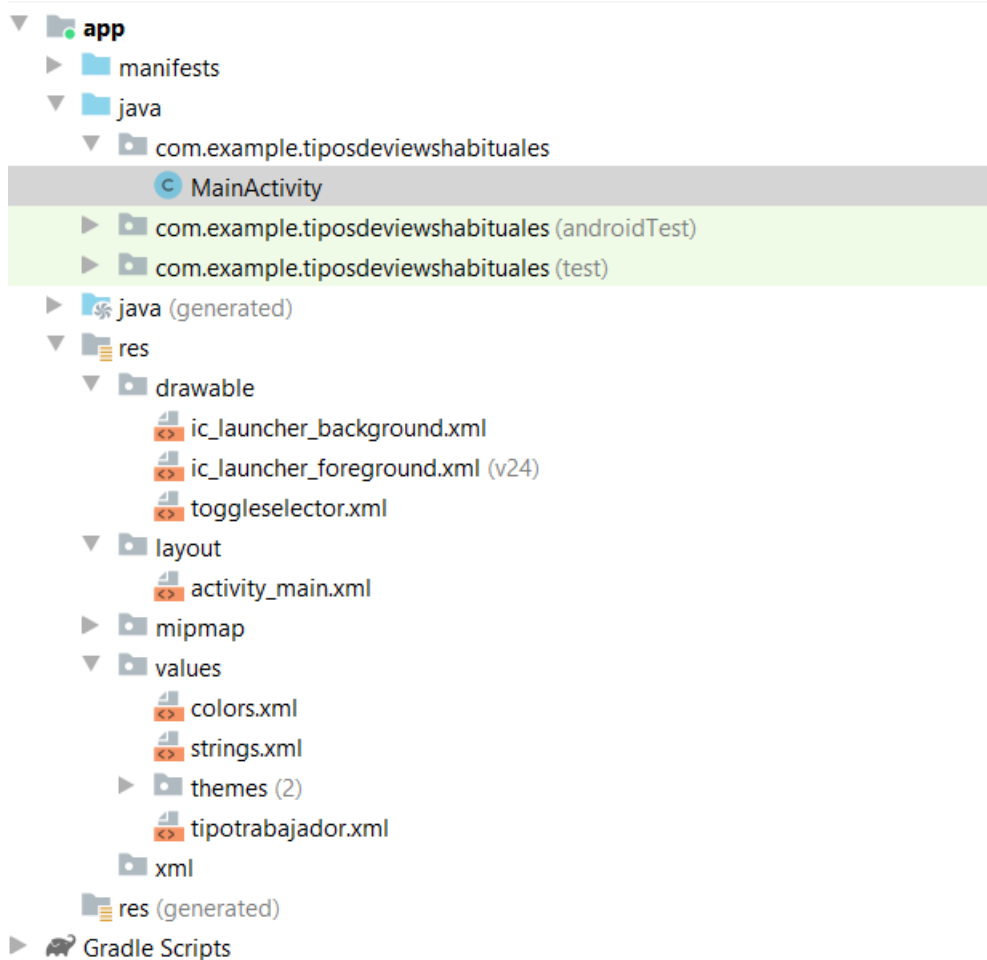
En este apartado podemos encontrar los elementos necesarios para **crear nuestros propios Views**.



Todos estos views cuentan con una serie de propiedades o atributos que definen su comportamiento, su colocación, su tamaño y el resto de propiedades.

Nota: Para los apuntes se proporciona el **proyecto TipoDeViewsHabituales**. Trabajaremos fundamentalmente con el Layout en el fichero `activity_main.xml`. La parte de código Java en el fichero `MainActivity.java`





2 Propiedades de los Views

Las **propiedades de los Views** son las características de cada uno de los elementos. Dentro de estas características podemos encontrar algunas que son útiles para algunos Views pero no son útiles para otros y otras características que serán comunes e importantes para los otros.

En cuanto las propiedades básicas podemos hablar de:

IDs - El identificador con el que haremos referencia a dicha vista o elemento.

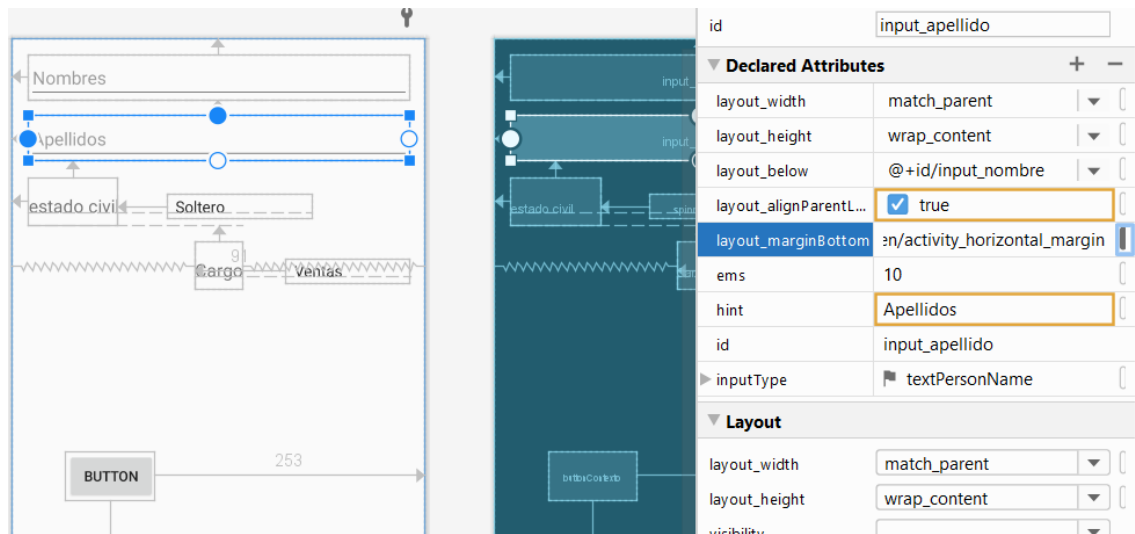
Dimensiones - El tamaño que tendrá el elemento dentro de la actividad.

Posiciones - Dónde estará colocado el elemento y respecto a qué lo estará.

Como ya hemos explicado anteriormente, la posición la podemos modificar simplemente arrastrando el View por la pantalla hasta la posición deseada. El tamaño se puede modificar igualmente seleccionando el elemento y arrastrando desde los cuadros que aparecen en el contorno del mismo.

Los IDs, también se pueden modificar haciendo doble clic sobre el elemento en cuestión, sin tener así que buscar su propiedad concreta.

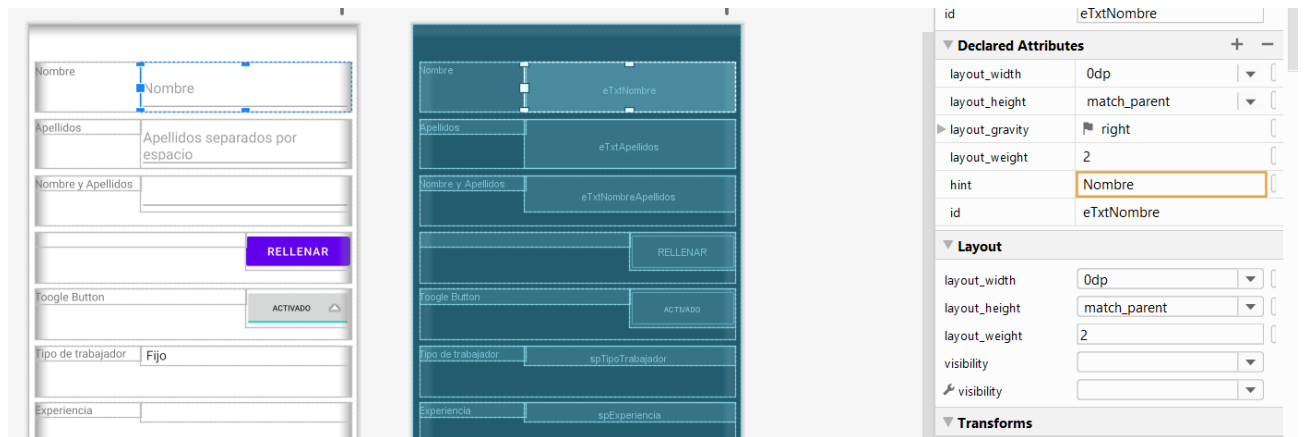
Lo podéis ver en el proyecto proyectoPartes para el layout ejemplo_relative_layout.xml



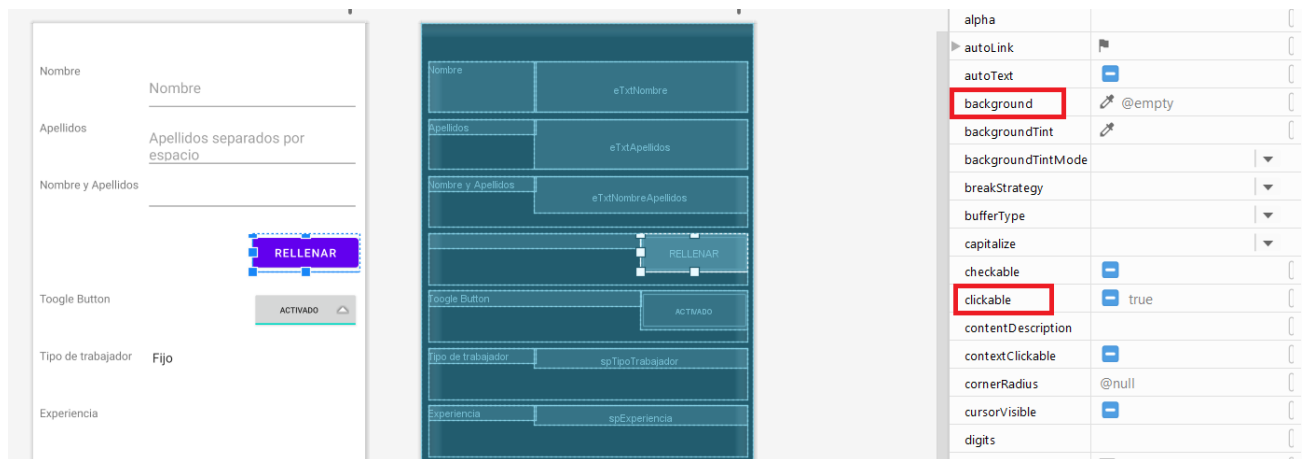
Antes de Android Studio, todos estos elementos debían modificarse desde el fichero .xml correspondiente a las vistas de los elementos, donde allí se especificaban estos y otros parámetros. Con Android Studio, Google ha **simplificado** notablemente esta tarea y, además de las formas anteriores, también nos permite configurar las propiedades de cada elemento desde el apartado concreto “propiedades” del IDE.

No vamos a entrar en todas las posibilidades de los elementos a las que podemos acceder desde aquí porque son muchas, sin embargo, sí que vamos a ver las **principales que tendremos** que tener en cuenta prácticamente siempre y, según vayamos viendo la necesidad de modificar otras, lo iremos indicando y completando en esta categoría.

- **Layout:width:** Anchura del elemento en cuestión.
 - **fill_parent, match_parent, wrap_content** son los valores mas habituales
- **Layout:height:** Altura del elemento en cuestión.
 - **fill_parent, match_parent, wrap_content** son los valores mas habituales
- **Layout:margin** - Margen exterior del elemento respecto a los demás.



- **Background:** Aspectos relacionados con el fondo del elemento.
- **Clickable:** Indica si el elemento se puede pulsar.



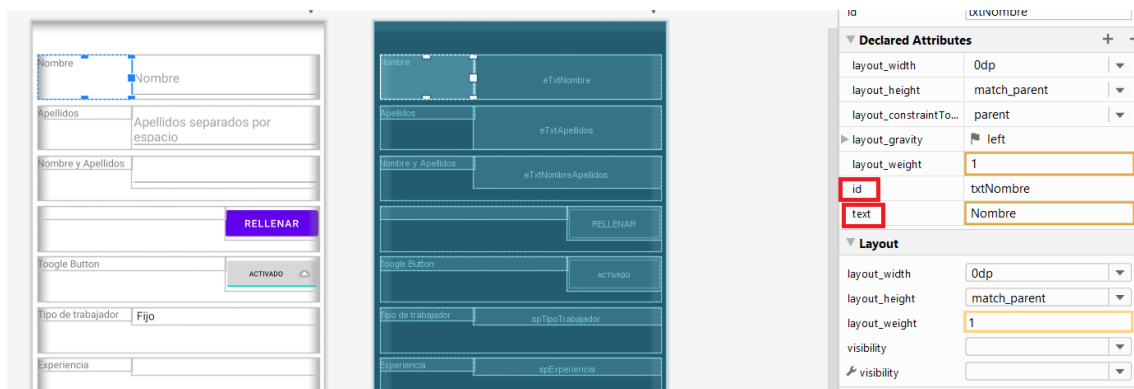
- **ID:** El identificador del elemento.
- **LongClickable:** Si la pulsación larga tiene alguna finalidad.
- **OnClick:** Qué hace cuando se le pulsa.
- **Padding:** Margen interior del elemento.
- **Rotation:** Nos permita rotar el elemento en la pantalla.
- **Scale:** Configuramos la escala.
- **Scrollbar:** Nos permite configurar la barra de desplazamiento.
- **Tag:** La etiqueta del elemento, útil para identificarlo más fácilmente en el desarrollo.
- **Text:** El texto que tiene el elemento.
- **Visibility:** Indica la visibilidad del elemento en la actividad.

3 Vistas de Texto

Estas vistas son los controles que nos sirven para mostrar y recoger información por pantalla. Son básicamente etiquetas y cajas de texto.

3.1 TextView Android

Como dijimos antes, el control **TextView** es un clásico en la programación de GUIs. Los **TextView** (etiquetas de texto) se utilizan como medio de salida, es decir, para mostrar un determinado texto al usuario. Al igual que en el caso de los botones, el texto del control se establece mediante la propiedad **android:text**. A parte de esta propiedad, las etiquetas que definen su tamaño son las mínimas que debemos definir (es muy parecido a los botones que ya vimos antes) se destacan las siguientes:



```
<TextView
    android:id="@+id/txtNombre"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:layout_gravity="left"
    android:layout_margin="4dp"
    android:padding="4dp"
    android:textSize="12dp"
    android:textColor="@color/black"
    android:typeface="sans"
    android:text="Nombre" />
```

Arriba podéis ver el resultado de la pantalla con el código xml que hemos programado voy a intentar explicar cada propiedad lo que hace para que poco a poco vallamos entendiendo cada línea ahora expliquemos cada elemento:

android:id. Esto ya lo vimos en el tema anterior, esta propiedad es el ID del control, con el que podremos identificarlo de forma única más tarde en nuestro código. Vemos que el identificador lo escribimos precedido de “@+id/”. Esto

tendrá como efecto que al compilarse el proyecto se genere automáticamente una nueva constante en la clase R para dicho control.

android:text. Texto del control. En Android, el texto de un control se puede especificar directamente, o bien utilizar alguna de las cadenas de texto definidas en los recursos del proyecto (archivo strings.xml), en cuyo caso indicaremos su identificador precedido del prefijo “@string/”.

android:layout_height y android:layout_width. Dimensiones del control con respecto al layout que lo contiene. Esta propiedad tomará normalmente los valores “wrap_content” para indicar que las dimensiones del control se ajustarán al contenido del mismo, o bien “fill_parent” para indicar que el ancho o el alto del control se ajustará al ancho o alto del layout contenedor respectivamente.

android:layout_margin. Este parametro indica el espacio entre el control (en este caso botón) y su padre en este caso el Layout.

android:padding. Este parámetro indica el espacio entre Texto o Imágenes que pongamos dentro del control (en este caso el botón) y el propio control.

android:gravity. Esta es la gravedad del control, la verdad es que es una forma original de poner la alineación, si entre las comillas de este parámetro pulsamos la famosa combinación de teclas Control+Espacio, Android Studio nos dará todos los parámetros que podemos usar aquí, si ponemos solo "center" se centrará su contenido tanto vertical como horizontal.

android:TextSize. Aquí indicamos un tamaño, si no ponemos esto, se selecciona un tamaño por defecto estándar, si te parece grande o pequeño ve probando usando la unidad dp que vimos en el tema anterior.

android:background. Con este parámetro definimos el color de fondo del control.

android:textColor. Color del texto.

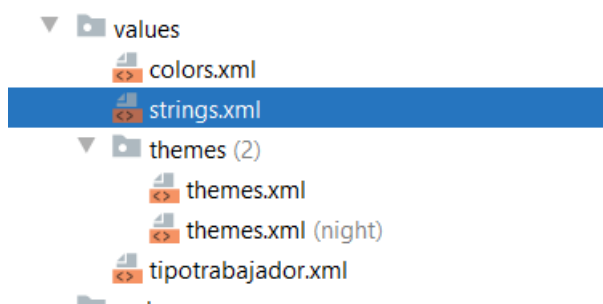
android:typeface. Estilo del texto, negrita, cursiva...

Bueno, hasta aquí con el XML del TextView, muchos de los parámetros que hemos visto son comunes con el siguiente control, asique no lo voy a repetir; P.D: Recuerda que en cualquier momento, puedes usar las combinaciones de

teclas Control+Espacio (no me cansare de recordarlo) no tienes que conocer todas las propiedades.

3.2 Cambiando los valores constantes con el fichero string.xml

Los apuntes están realizados de la manera más sencilla posible pero en este punto vamos a introducir el uso para las cadenas de Texto de TextViews y otros componentes el fichero de recursos de valores strings.xml.



Lo encontrareis con esta información, lo habitual es rellenarlo con valores constantes que aparecen en nuestra pantalla, es muy útil para hacer aplicaciones multilinguaje.

```
<resources>
  <string name="app_name">TiposDeViewsHabituales</string>

</resources>
```

Vamos a añadir para el textView anterior su texto. Añadid esta linea al XML.

```
<string name ="txtNombre">Nombre:</string>

<resources>
  <string name="app_name">TiposDeViewsHabituales</string>
  <string name ="txtNombre">Nombre:</string>

</resources>
```

Y cambia en el TextView txtNombre el texto por lo siguiente:

```
android:text="@string/txtNombre"
```

```
<TextView
    android:id="@+id/txtNombre"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:layout_gravity="left"
    android:layout_margin="4dp"
    android:padding="4dp"
    android:textSize="12dp"
    android:textColor="@color/black"
    android:typeface="sans"
    android:text="@string/txtNombre" />
```

De esta manera cógera el texto del fichero String. Podemos reutilizar texto, y no haría falta modificarlo en cada layout que tengamos, sólo modificarlo en el fichero strings.xml.

3.2.1 Actividad String.xml

- 1) Cambiad los textos de los TextViews que aparecen en activity_main.xml para que se recojan en el fichero strings.xml en vez de ser cadenas constantes.
- 2) Cambiad los atributos hint de los Edittext para que sean recogidos en el fichero strings.xml

3.3 EditText Android

El control **EditText** (caja de edición de texto) es otro componente indispensable de casi toda aplicación Android. Es uno de los medios que tiene el usuario para introducir datos a la aplicación. Poco más os puedo decir de este control... os pongo un ejemplo:

```
<EditText
    android:id="@+id/eTxtNombre"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_gravity="right"
    android:hint="Nombre"
    android:inputType="textPersonName"
    android:layout_weight="2" />
```

Aparte de los atributos que comparte con el `TextView`, para este control tenemos algunos parámetros que son exclusivos, los más utilizados son estos dos:

android:hint. Con este parámetro pondremos un texto por defecto mientras el campo este vacío, es decir, el lo que ve el usuario antes de que el escriba nada.

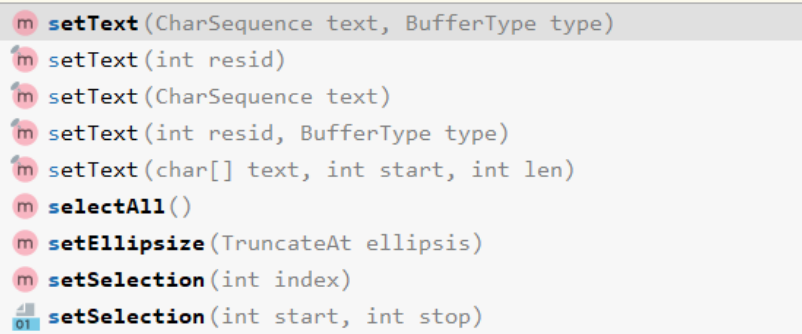
android:inputType. Con este parámetro indicamos el tipo de teclado que quieres que aparezca cuando pulsas sobre el campo, si por ejemplo solo vamos a introducir números, se puede indicar que aparezca un teclado numérico. Como siempre cuando te coloques entre las comillas pulsa `Control+Espacio` para ver las opciones que puedes poner para este parámetro.

3.3.1 Código Java para los `TextView` y `EditText`

Estos controles son más sencillos de gestionar. Como siempre si queremos actuar sobre los controles, tendremos que localizar los controles en nuestro código Java y asignarlos a una variable y a partir de ahí podremos hacer cosas con ellos (cambiar colores, textos, tamaños, leer, escribir...) pongamos un ejemplo con las dos cosas básicas:

```
@Override
public void onClick(View v) {

    if ( !nombre.getText().toString().equals("") &&
        !apellidos.getText().toString().equals("") ) {
        nombreApellidos.s...
        nombreApelli...
    } else {
        Toast.makeText...
        Toas...
        .sho...
    }
}
```



Recuerda, que cuando escribimos la variable podemos aprovecharnos de la ayuda de Android Studio para ver que cosas podemos hacer con estos controles, una vez más estoy usando la famosísima combinación de teclas **control+espacio** para ver los métodos que puedo asignar a la variable `mitexto` que es un `TextView`.

Vamos a hacer un ejercicio divertido, escribiremos algo en la caja **EditText** y cuando pulsemos un botón, este texto lo asignaremos al control **TextView**.

Antes de nada, voy a explicar un concepto nuevo, **declarar variable GLOBALES para los componente** sen el proyecto. **Se puede hacer locales también**. Hay 2 tipos de variables, las locales y las globales, la diferencia entre cada una de ellas es que las globales las podemos utilizar en cualquier parte de nuestro programa, y las locales solo en los métodos (partes del programa) donde hayan sido definidas, a partir de ahora las variables de los controles las asignaremos de forma global, para poder utilizarlas desde cualquier sitio. **Las variables Globales se asignan antes del onCreate** de la **Actividad**, la asignación de variables en cualquier otro sitio de nuestro programa se considera Locales, si estas variables las quieres usar en otras partes es probable que te diga que la variable no exista, entonces deberás de definirla de forma global, observa, como detalle muy importante, que las variables globales son de color azul.

MainActivity.java

```
import com.google.android.material.snackbar.Snackbar;
```

```
public class MainActivity extends AppCompatActivity {  
  
    private EditText nombre, apellidos, nombreApellidos;  
    private Button btnRellenar;  
    private ToggleButton toggleButton;  
    private Spinner spTipoTrabajador, spExperiencia;  
  
    private String[] arrayExperiencia= {"NOVATO", "JUNIOR", "SENIOR", "VETERANO"};
```

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    nombre= findViewById(R.id.eTxtNombre);  
    apellidos = findViewById(R.id.eTxtApellidos);  
    nombreApellidos = findViewById(R.id.eTxtNombreApellidos);  
    btnRellenar = findViewById(R.id.btnRellenar);  
    toggleButton = findViewById(R.id.btnToggle);  
    spTipoTrabajador= findViewById(R.id.spTipoTrabajador);
```

Fijaos como para recoger el componente en una variable java usamos el método **findViewById** de la clase Activity. Este nos carga el componente XML en una variable Java. **R.id.eTxtNombre** es el identificado que le hemos asignado en el

XML. R es un fichero de constantes de tipo long que asocia un numero a cada identificador que añadimos en los XML Android. Antes se podía acceder al fichero desde el ID. En la actualidad se genera en tiempo de ejecución.

```
private EditText nombre;  
  
nombre= findViewById(R.id.eTxtNombre);
```

Dentro de `activity_main.xml` encontrareis el EditText declarado como XML

```
<EditText  
    android:id="@+id/eTxtNombre"  
    android:layout_width="0dp"
```

4 Los Botones en Android

En el apartado anterior hemos visto los distintos tipos de layout con los que contamos en Android para distribuir los controles de la interfaz por la pantalla del dispositivo. En las próximas lecciones vamos a hacer un recorrido por los diferentes controles, y en esta concretamente los de tipo **Button (Botón)** que pone a nuestra disposición la plataforma de desarrollo Android.

Vamos a ver los diferentes tipos y cómo podemos personalizarlos, Android en este caso nos proporciona tres tipos de botones:

Button: Es el típico botón normal, contiene un texto y puede contener imágenes de tipo icono que puedes alinear con el texto.
ToggleButton: es de tipo on/off, contiene una rayita que indica ese on/off.
ImageButton: El botón contiene una imagen, es muy útil para cuando queremos hacer cosas vistosas y de diseño con formas raras, ya que el Button es rectangular, y aquí podemos usar cualquier forma poniendo el fondo transparente o blanco.

Nombre

Apellidos

Nombre y Apellidos

RELLENAR

Toggle Button

ACTIVADO

Tipo de trabajador Aut3nomo

Experiencia NOVATO

Boton normal

Boton Toggle

4.1 Ejemplo Button Android

Para programar un bot3n necesitamos 2 cosas: por un lado, todas las **propiedades del bot3n** que lo programamos en el xml (tama3o, color, texto del bot3n, alineaci3n, márgenes...) y por el otro lo que llamamos el Listener que es el encargado de "escuchar" en nuestro c3digo cuando es pulsado. Vamos a analizar la parte del XML.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_marginTop="8dp"
    android:layout_weight="1"
    android:orientation="horizontal"
    android:weightSum="3">
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text=""
        android:layout_weight="2"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_gravity="left"/>
    <Button
        android:id="@+id/btnRellenar"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:layout_weight="1"
        android:background="@color/blue"
        android:text="Rellenar" />
```

Arriba vemos el resultado de la pantalla con el código xml que hemos programado y vamos a **intentar explicar cada propiedad** lo que hace para que poco a poco vallamos entendiendo cada línea, antes de nada vamos a reforzar lo que vimos en los aountes anteriores de Layouts. Fíjate que lo que es etiquetado el **weight** como "1" es el **Layout que lo contieen**, es decir, el **contenedor**, con las propiedades de tamaño (ancho y alto) wrap content , en **height**, para que despliegue el contenido del botón, y en Width 0 para que se ajuste al peso del linear layout padre como vimos en los apuntes de Layouts. El **WeightSum** es 3 para el **LinearLayout** con lo que ocupará una tercera parte del **LinearLayOut** de orientación horizontal. También le hemos **puesto la propiedad layout_gravity="right"** coloca el botón a la derecha del linear que lo contiene.

android:id. Este atributo ya lo vimos en el tema anterior, **esta propiedad es el ID del control, con el que podremos identificar el control** de forma única más tarde en nuestro código. Vemos que el identificador lo escribimos precedido de “@+id/”. Esto tendrá como efecto que al compilarse el proyecto se genere automáticamente una nueva constante en la clase R para dicho control.

android:text. Texto del control. En Android, el **texto de un control se puede especificar directamente**, o bien utilizar alguna de las **cadenas de texto definidas en los recursos del proyecto** (archivo strings.xml), en cuyo caso indicaremos su identificador precedido del prefijo “@string/”.

android:layout_height y android:layout_width. Dimensiones del control con respecto al layout que lo contiene. Esta propiedad tomará normalmente los valores “wrap_content” para indicar que las dimensiones del control se ajustarán al contenido del mismo, o bien “fill_parent” para indicar que el ancho o el alto del control se ajustará al ancho o alto del layout contenedor respectivamente.

android:layout_margin. Este parámetro indica el espacio entre el control (en este caso botón) y su padre en este caso el Layout.

android:padding. Este parámetro indica el espacio entre Texto o Imágenes que pongamos dentro del control (en este caso el botón) y el propio control.

android:gravity. Esta es la gravedad del control, la verdad es que es una forma original de poner la alineación, si ponemos solo “center” se centrará su contenido tanto vertical como horizontal.

android:drawable. Con este parámetro pondremos una imagen al botón, que no es lo mismo que el ImageButton, que todo el botón es en si una imagen, con esto ponemos una especie de icono dentro del botón, por ejemplo si queremos poner el tipo Play, Stop las flechas de avance o retroceso... existen variantes de este parámetro para poner la imagen a la derecha o izquierda, arriba o abajo.

android:Text. Con este parámetro definimos el texto del botón, podemos usar un recurso R usando “@String/texto o directamente un texto entre comillas.

android:TextSize. Ahi indicamos un tamaño, si no ponemos esto, se selecciona un tamaño por defecto standard, si te parece grande o pequeño ve probando usando la unidad dp que vimos en el tema anterior.

android:background. Con este parámetro definimos el color de fondo del Botón.

android:OnClick. Esta es una propiedad que nos ayuda con el Listener, al final de la lección hablo de esto, que es la parte Java del Botón, esto nos facilita la vida, lo malo es que solo se puede utilizar a partir de la API 6 de Android, no creo que vayáis a hacer un proyecto con esta API yo uso esta opción, es menos lioso. **Esta en desuso en la actualidad**

Bueno, hasta aquí con el XML del Botón, muchos de los parámetros que hemos visto son comunes con los otros 2 tipos de botones, recuerda que en cualquier momento, puedes usar las combinaciones de teclas Control+Espacio no tienes que conocer todas las propiedades, Android Studio nos ayuda a esto, úsalo.

```
<Button
    android:id="@+id/btnRellenar"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    android:layout_weight="1"
    android:background="@color/blue"
    android:text="Rellenar"
/>
```

4.2 Ejemplo ToggleButton Android

Después de lo anterior, solo voy a decir las diferencias entre cada tipo, así que este es un tipo de botón que puede permanecer en dos estados, pulsado/no_pulsado. En este caso, en vez de definir un sólo texto para el control definiremos dos, dependiendo de su estado, asignamos estas propiedades, **android:textOn** y **android:textOff** para definir ambos textos, veamos un ejemplo básico de **ToggleButton**, piensa que puedes añadir más propiedades como el caso anterior. Yo personalmente no he usado estos controles nunca, prefiero usar el **ImageButton**, piensa que puedes cambiar las imágenes del botón luego en el programa cuando te convenga.

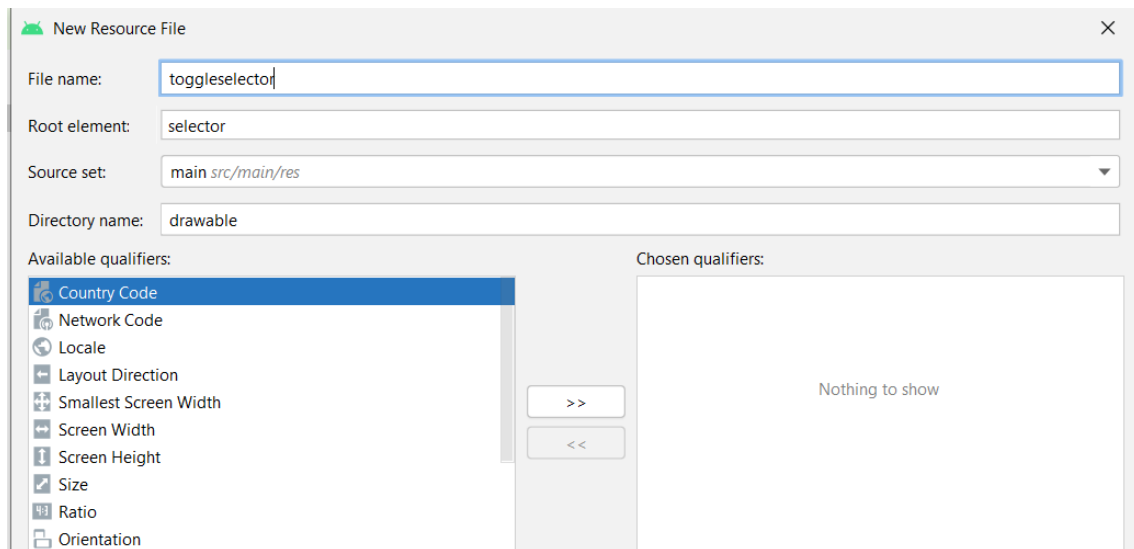
```
<ToggleButton
    android:id="@+id/btnToggle"
    android:drawableRight="@drawable/toggleselector"
    android:textSize="10sp"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    android:layout_weight="1"
    android:checked="true"
    android:textOff="Desactivado"
    android:textOn="Activado"/>
```

Para el texto las propiedades textOff, mostrará este texto cuando este desactivado (checked=false) y textOn para el caso contrario.

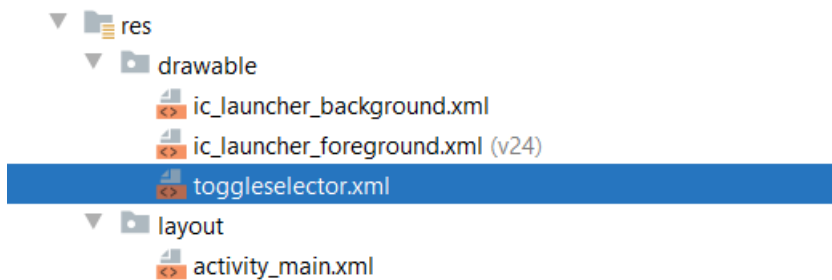
```
android:textOff="Desactivado"
    android:textOn="Activado"/>
```

Para la imagen que aparece en el toggle button definimos un drawable selector en la carpeta drawable. Se hace botón derecho con la carpeta drawable seleccionada, New-> Drawable Resource File

```
android:drawableRight="@drawable/toggleselector"
```



Podréis encontrar en el proyecto en la carpeta resources/drawable con nombre toggleselector.xml



```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable = "@android:drawable/arrow_up_float"
        android:state_checked="true"/>
    <item android:drawable="@android:drawable/arrow_down_float"
        android:state_checked="false"/>
</selector>
```

La flecha hacia arriba para cuando el botón está checked (activado) y la flecha hacia abajo cuando esta desactivado.

4.2.1 Practica. Intentar añadir un ImageButton como el siguiente

Intentareis ahora trabajar por vuestra cuenta. Añadid un ImageButton al proyecto. Para ello tendréis que copiar la siguiente imagen y añadirla a la carpeta drawable. Se puede copiar la imagen desde el explorador de Windows en la carpeta directamente. Podéis crear la imagen con Paint. Llamarla

emergenciastop.jpg. Para referenciar la imagen desde el XML del **ImageButton** lo podéis hacer como tenéis debajo en el código de ejemplo, `Android:src="@drawable/nombredelaimagen"`

Para referenciar la imagen desde



Es un Botón muy usado en cuanto queremos hacer cositas más vistosas, la principal diferencia es que en un **ImageButton** definimos una imagen a mostrar en vez de un texto. Esta imagen la definimos con la propiedad **android:src**. Normalmente asignaremos esta propiedad un recurso **Drawable** que hayamos incluido en la carpeta **/res/drawable**. Aquí os dejo un ejemplo de una de mis aplicaciones, use un **ImageButton** para hacer un Botón que pareciera un paro de emergencia, fíjate en las propiedades del xml, fíjate que solo hay 2 diferentes: **android:src** y **android:ContentDescription** esta última es un texto de descripción del botón a nivel informativo, ya que no aparece en nuestro programa por ningún sitio.

Podéis copiar esta imagen y añadirla a la carpeta drawable

```
<ImageButton
    android:id="@+id/center2"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_margin="0dp"
    android:layout_weight="1"
    android:background="@android:color/white"
    android:padding="0dp"
    android:scaleType="centerInside"
    android:src="@drawable/emergenciastop"
    android:contentDescription="Boton central" />
```

Bueno y hasta aquí la parte de xml, ahora explicaremos la parte de Java, para los botones.

4.3 Listener para Button de Android en el código Java

Voy a comenzar por la parte tradicional, una vez en nuestro Java, para usar el control Botón tenemos que localizarlo y definir la variable, después crearemos lo que se dice Listener que es el encargado de escuchar al botón que le dirá a nuestro Java cuando el usuario lo ha pulsado, esta parte podría ser parecido a los tres tipos de Botones, solo cambiando el tipo de variable (Button, ImageButton o ToggleButton).

Para añadir un listener Onclick en Android sobre un botón, usamos el método `setOnClickListener`. Creamos el objeto Listener `new View.OnClickListener()` {} y dentro declaramos su cuerpo. En este caso vamos a escribir lo que hay en el EditText `eTxtNombre` y en el EditText `eTxtApellidos` en el EditText `etxtNombreApellidos`, si están rellenos. Si no mandamos un mensaje de error de duración dos segundos con la clase `Toast`.

Importante en cada método de Listener que sobreescribimos recibimos como parámetro como mínimo la vista que genera el evento `onClick(View v)`. Podemos recibir información extra como posiciones en una lista, lista de elementos en Listener más complejos como veremos más adelante.

```
//Añadiendo un listener al boton
btnRellenar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        if ( !nombre.getText().toString().equals("") &&
            !apellidos.getText().toString().equals("") ) {

            nombreApellidos.setText(nombre.getText().toString()+ " " +
            apellidos.getText().toString());

        } else {

            Toast.makeText(MainActivity.this,"Rellena los campos nombre y
            apellidos para hacer funcionar el botón",
            Toast.LENGTH_LONG)
            .show();

        }

    }
});
```


The screenshot shows a mobile application interface with the following elements:

- A text input field labeled "Nombre" containing the text "Lolito".
- A text input field labeled "Apellidos" containing the text "Fernandez Lopez".
- A text input field labeled "Nombre y Apellidos" containing the text "Lolito Fernandez Lopez".
- A purple button with the text "RELENAR" (highlighted with a red rectangle).
- A "Toggle Button" at the bottom, which is currently in the "ACTIVADO" (Activated) state, indicated by a grey background and a checkmark icon.

Como veis en el ejemplo de arriba, explicado puramente desde Java lo que estamos haciendo para crear el Listener, es crearlo como una clase anónima a partir del interfaz `OnClickListener`, sobrescribiendo el método `onClick`

```
btnRellenar.setOnClickListener(new View.OnClickListener() {
```

4.4 Listener para Toggle Button

En este caso es igual que el anterior pero tenemos que estamos añadiendo el Listener para el toggle Button.

```
// Listener para toggle button

toggleButton.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        ToggleButton toggle = (ToggleButton) buttonView;

        if (toggle.isChecked()) {
            Toast.makeText(MainActivity.this,
                " El toggle button se ha activado
", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(MainActivity.this,
                " El toggle button se ha desactivado
", Toast.LENGTH_SHORT).show();
        }
    }
});
```

El método `setOnCheckedChangeListener` es el que nos permite añadir el Listener para **comprobar cuando el botón cambia de estado**. Hay mas botones compuestos en Android además del Toggle Button. Por eso casteamos

```
CompoundButton buttonView
```

```
ToggleButton toggle = (ToggleButton) buttonView;
```

El método `toggle.isChecked()` nos permite comprobar si el botón ha sido activado.

4.5 Actividad.

1. Crear un Listener para el ImageButton que has añadido.
2. Que muestre con un mensaje de tipo Toast cuando el botón haya sido pulsado.

5 Spinner

El **spinner** es una view que actúa como una lista desplegable en Android studio. Puede cargarse estáticamente a través del fichero `arrays.xml` de la carpeta `value` dentro de la sección `resources` en el proyecto, o dinámicamente a través de un adapter. En este apartado veremos como hacerlo.

5.1 Creando el Spinner

Vamos **a crear dos Spinners para este ejemplo**. Este **primero para Tipo de trabajador para el que cargaremos sus datos desde un XML**. Con la propiedad `entries` indicamos el xml donde están los datos que será de tipo `values` y `arraystring`. Lo podéis **encontrar en la carpeta de resources/values** con nombre `tipotrabajador.xml`. Mas adelante os indico como podríais crearlo vosotros mismos.

```
android:entries="@array/tipotrabajador"
```

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"
```

```

    android:weightSum="3"
    android:orientation="horizontal"
    android:layout_marginTop="8dp">

    <TextView
        android:id="@+id/TxtTipoTrabajador"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Tipo de trabajador"
        android:layout_weight="1"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_gravity="left"/>

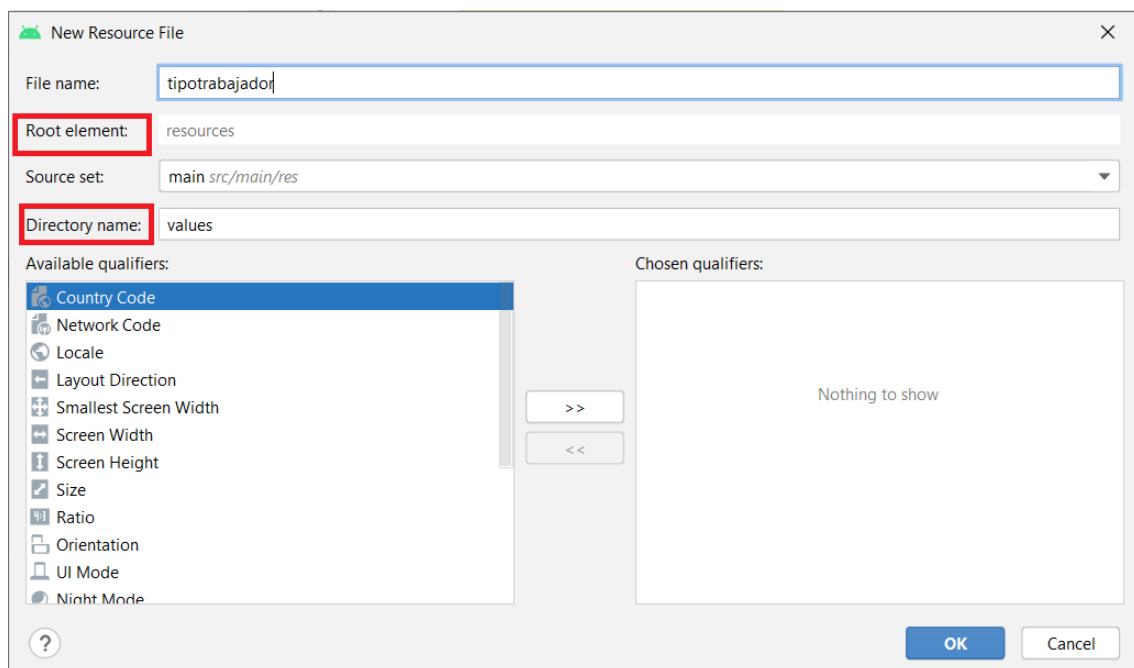
    <Spinner
        android:id="@+id/spTipoTrabajador"
        android:entries="@array/tipotrabajador"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:layout_gravity="right"/>

</LinearLayout>

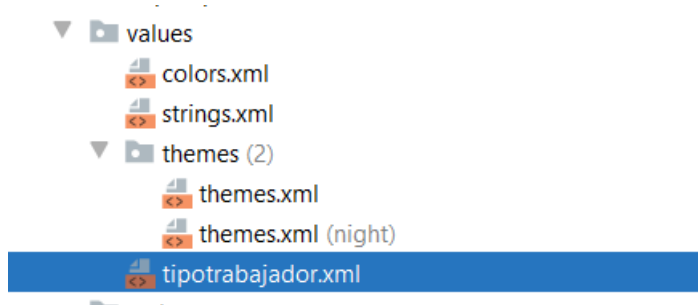
```

Una vez que hemos creado el spinner, necesitaremos rellenar el spinner con tipo de trabajador que queramos, para eso vamos a crear un recurso string-array en XML en el que indiquemos todos los tipos de trabajador.

Botón derecho sobre la carpeta recursos y new-> Android resource file. El fichero será de tipo resources e irá al directorio values.



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="tipotrabajador">
        <item>Fijo</item>
        <item>Temporal</item>
        <item>Autónomo</item>
    </string-array>
</resources>
```



La ventaja que ofrece la primera forma es que, al tener el recurso previamente configurado, **se puede asignar directamente en el XML** del layout al propio Spinner, sin necesidad de código extra en Java. Para ello utilizaríamos el atributo `android:entries=""` y le daremos como valor el nombre que asignamos al string-array que contiene los tipos de trabajador. En el caso de nuestro ejemplo sería `android:entries="@array/tipotrabajador"`.

```
<TextView
    android:id="@+id/TxtTipoTrabajador"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Tipo de trabajador"
    android:layout_weight="1"
    app:layout_constraintTop_toTopOf="parent"
    android:layout_gravity="left"/>
```

Sin embargo, **para utilizar la colección desde el código Java tendremos que crear un Adapter** (una clase que sirve de puente entre el Spinner y la colección de datos que contiene, y que se encarga de crear las vistas necesarias para mostrar los datos). Este tipo de objeto lo explicaremos más adelante, cuando mostremos cómo mostrar vistas personalizadas para estos datos (por ejemplo, al lado del nombre del animal se podría poner una imagen).

Cada opción tiene una utilidad distinta: Si son **opciones fijas** utilizaremos un recurso XML que nos ahorrará trabajo. Si son **opciones dinámicas** (datos introducidos en una Base de Datos por ejemplo), **la tendremos que gestionar obligatoriamente a través del código Java.**

5.2 Introduciendo datos en el Spinner desde Java con un array

La primera forma de rellenar el Spinner (a partir del XML) ya la hemos explicado, falta explicar cómo hacerlo desde el código Java. Para rellenar el Spinner en primer lugar crearemos el adapter que hemos mencionado, utilizaremos el **ArrayAdapter** para esto. Esta clase tiene varios constructores, que se podrán ver en Android Developer. En este ejemplo utilizaremos el constructor que utiliza un Context (el propio de la Activity), un recurso para mostrar datos (la vista, utilizamos una por defecto de Android) y el array de String que hemos creado:

Tenemos el array:

```
private String[] arrayExperiencia= {"NOVATO", "JUNIOR", "SENIOR",  
"VETERANO"};
```

Una variable de tipo Spinner:

```
spExperiencia = findViewById(R.id.spExperiencia);
```

Ahora tenemos que **asignar el adapter que hemos creado al Spinner que tenemos**, para eso **declararemos una variable de Spinner**, que inicializaremos apuntando al Spinner que hemos creado en el layout, posteriormente le asignaremos el adapter al Spinner:

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,  
    android.R.layout.simple_spinner_dropdown_item, arrayExperiencia);
```

```
spExperiencia.setAdapter(adapter);
```

Con esto ya **hemos creado nuestros Spinner**, y hemos rellenado los datos para que se muestren al desplegarlos.

The screenshot shows a web form titled "TiposDeViewsHabituales". It contains three input fields: "Nombre", "Apellidos", and "Nombre y Apellidos". Below these fields is a purple button labeled "RELLENAR". Further down, there are two dropdown menus. The first is labeled "Tipo de trabajador" and has "Fijo" selected. The second is labeled "Experiencia" and has a dropdown menu open showing four options: "NOVATO", "JUNIOR", "SENIOR", and "VETERANO".

5.2.1 Actividad de Spinner

Investigar en la web como cargar una colección en un Spinner. Pasos:

1. Cambiar el array de experiencia por una colección List `listExperiencia` con los mismos datos
2. Modificar el código para cargar los datos en el Spinner desde la colección.

5.3 Seleccionando el cambio de opción de un spinner

Para finalizar vamos a explicar como añadir a un Spinner un Listener para controlar los cambios en dicho Spinner. Una posible opción es la siguiente, añadir un listener `OnItemSelectedListener`, como es habitual en Android y no hará falta explicar de nuevo. Sólo cabe resaltar que para coger el Texto del elemento seleccionado, debemos acceder a la vista del adapter `AdapterView<?> parent` y coger el elemento a partir de la posición. En los propios parámetros del método `onItemSelected` están las herramientas para poder acceder al elemento seleccionado. Por un lado, el `AdapterView` con toda la lista de elementos por otro lado, la posición.

```
parent.getItemAtPosition(position).toString()
```

```
spTipoTrabajador.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int
position, long id) {
        Toast.makeText(MainActivity.this,
            " El elemento seleccionado para el spinner tipo trabajador es
"+
parent.getItemAtPosition(position).toString(), Toast.LENGTH_SHORT).show();

    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {

    }
});
```