



cpi'fp

Los Enlaces

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

2º DESARROLLO DE APLICACIONES MULTIPLATAFORMA

TEMA 2. INTERACCIÓN CON ELEMENTOS

0. TIPOS DE ELEMENTOS

Hasta ahora hemos visto los elementos básicos:

- View
- Button
- TextView
- EditText

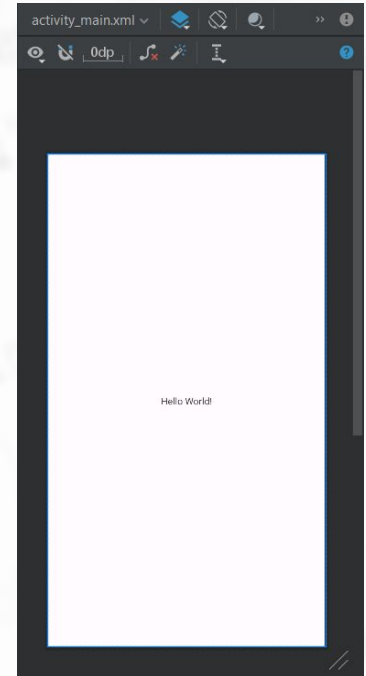
Éstos nos sirven para crear aplicaciones sencillas.
Vamos a ver cómo podemos interactuar con ellos.



0. TIPOS DE ELEMENTOS

Al crear un nuevo proyecto en *AndroidStudio* con una *Empty Views Activity* la pantalla inicial muestra el mensaje *Hello World!* en un *TextView*:

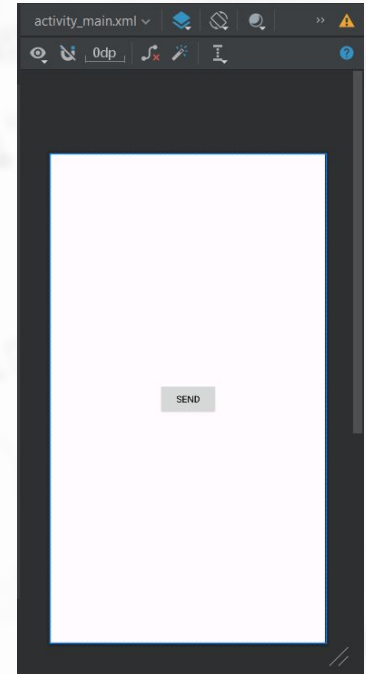
```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"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



0. TIPOS DE ELEMENTOS

Podemos cambiar el tipo de elemento para que sea un *Button* y cambiarle el texto a *Send*. La propiedad *wrap_content* ajusta el tamaño de la View al contenido:

```
<androidx.appcompat.widget.AppCompatButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Send"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```



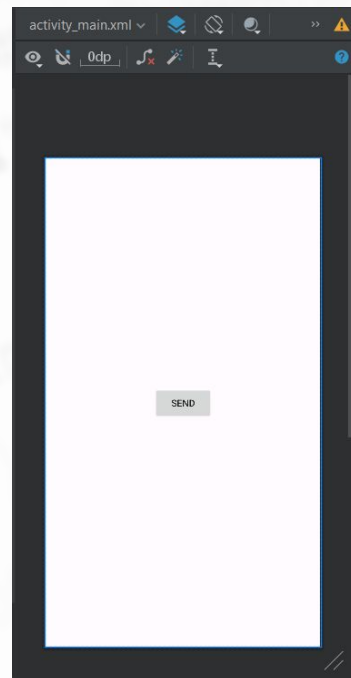
0. TIPOS DE ELEMENTOS

Podemos seleccionar la opción Button del desplegable o `androidx.appcompat.widget.AppCompatButton`. Consultando la documentación:

An `AppCompatButton` is simply a [Button](#) which supports compatible features on older versions of the platform, including:

- Allows dynamic tint of its background via the background tint methods in [ViewCompat](#).
- Allows setting of the background tint using [R.attr.backgroundTint](#) and [R.attr.backgroundTintMode](#).
- Allows setting of the font family using [R.attr.fontFamily](#)

This will automatically be used when you use `Button` in your layouts and the top-level activity / dialog is provided by `appcompat`. You should only need to manually use this class when writing custom views.



0. TIPOS DE ELEMENTOS

Como no vamos a hacer uso de estos métodos, simplemente seleccionaremos la opción Button y le daremos una *id*:

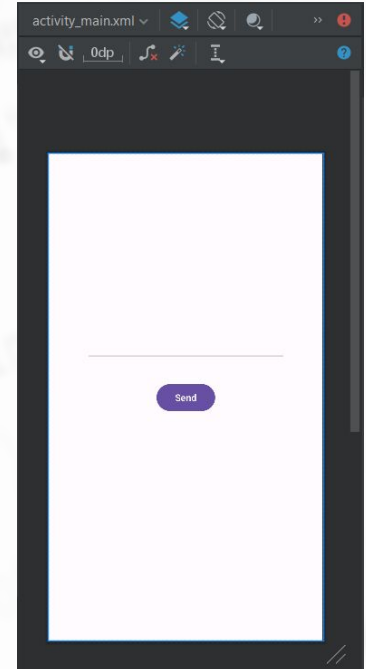
```
<Button
    android:id="@+id/buttonSend"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Send"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



0. TIPOS DE ELEMENTOS

Incluimos un `EditText` que situamos encima del botón creado y le asignamos `id`. Si le ponemos menos de 48dp de altura, *Android Studio* nos dirá que es muy pequeño:

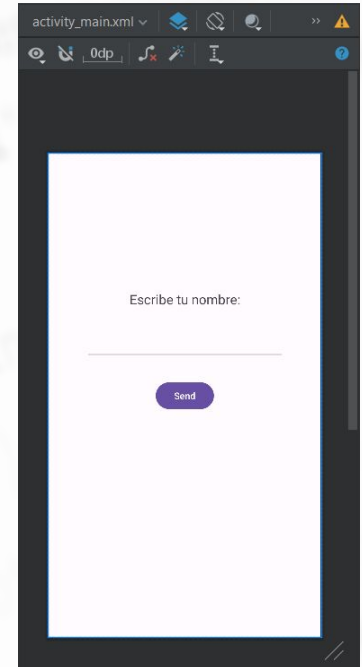
```
<EditText
    android:id="@+id/etName"
    android:layout_width="300dp"
    android:layout_height="48dp"
    android:layout_marginBottom="30dp"
    android:textAlignment="center"
    android:maxLines="1"    <!--Vale también singleLine="true"-->
    app:layout_constraintBottom_toTopOf="@id/buttonSend"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />
```



0. TIPOS DE ELEMENTOS

Ahora un `TextView` con el texto *Escribe tu nombre* que situamos encima del botón creado y le asignamos *id*:

```
<TextView
    android:id="@+id/textName"
    android:layout_width="300dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="30dp"
    android:labelFor="@id/etName"
    android:text="@string/escrive_tu_nombre"
    android:textAlignment="center"
    android:textSize="20sp"
    app:layout_constraintBottom_toTopOf="@id/etName"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />
```

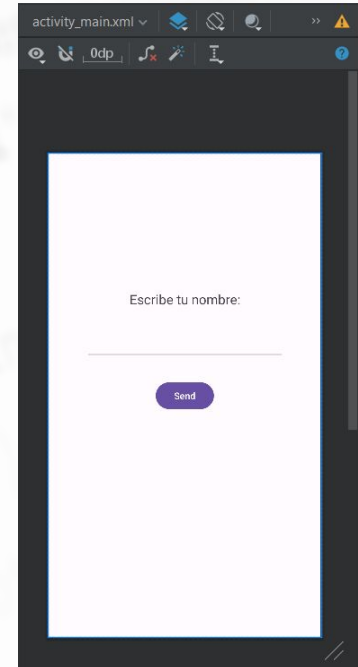


0. TIPOS DE ELEMENTOS

Warnings (advertencias):

- **Hardcoded text.** Nos indica que podemos almacenar nuestros textos como referencias en el archivo *strings.xml* de la carpeta */res/values*.
- **Missing inputType.** Debería incluirse un atributo con el tipo de datos que se van a leer en el *EditText*.
- **Use Autofill.** Nos sugiere incluir un atributo llamado *autofillHints* para las reglas de autollenado de datos.

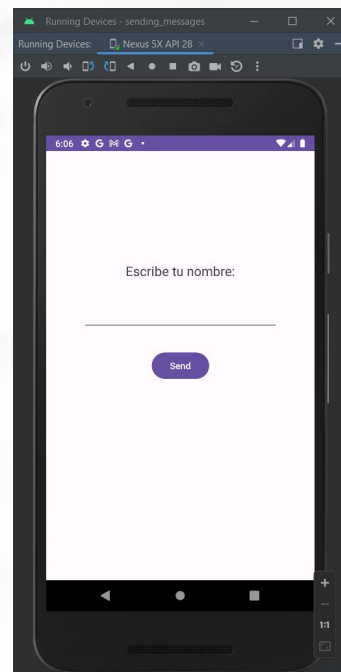
En general, nos preocuparemos solo de los errores de código y omitiremos estos warnings.



1. INTERACCIONES

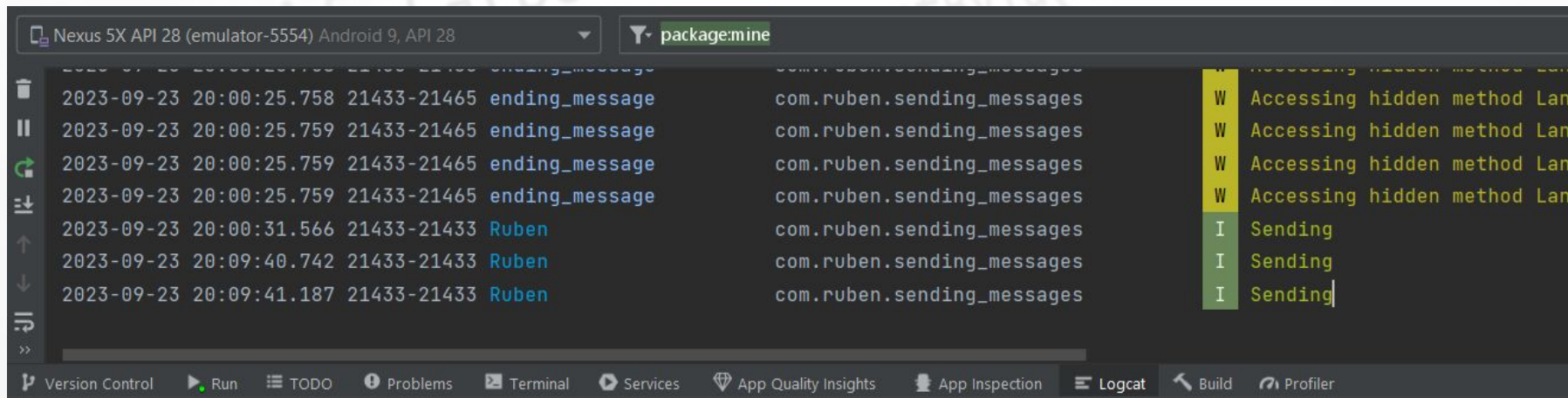
Haremos uso del comando *findViewById* y el método *setOnClickListener*:

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        //Al arrancar la pantalla  
  
        var btnSend = findViewById<Button>(R.id.buttonSend)  
        btnSend.setOnClickListener {  
            Log.i("Ruben", "Sending")  
        }  
    }  
}
```



1. INTERACCIONES

Al ejecutar la aplicación en nuestro dispositivo virtual y pulsar el botón, podremos ver en la consola (*Logcat*) las pulsaciones donde se muestran la etiqueta y el mensaje que hayamos definido en el comando *Log.i()*:



```
2023-09-23 20:00:25.758 21433-21465 ending_message com.ruben.sending_messages W Accessing hidden method Lan
2023-09-23 20:00:25.759 21433-21465 ending_message com.ruben.sending_messages W Accessing hidden method Lan
2023-09-23 20:00:25.759 21433-21465 ending_message com.ruben.sending_messages W Accessing hidden method Lan
2023-09-23 20:00:25.759 21433-21465 ending_message com.ruben.sending_messages W Accessing hidden method Lan
2023-09-23 20:00:31.566 21433-21433 Ruben com.ruben.sending_messages I Sending
2023-09-23 20:09:40.742 21433-21433 Ruben com.ruben.sending_messages I Sending
2023-09-23 20:09:41.187 21433-21433 Ruben com.ruben.sending_messages I Sending
```

1. INTERACCIONES

Creamos otra variable para el EditText y otra que almacene el texto que el usuario escriba en él:

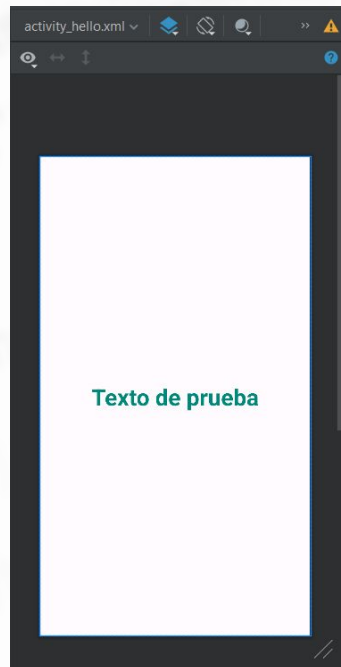
```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        var btnSend = findViewById<Button>(R.id.buttonSend)  
        var userText = findViewById<EditText>(R.id.etName)  
  
        btnSend.setOnClickListener{  
            var name = userText.text.toString()  
        }  
    }  
}
```



1. INTERACCIONES

Ahora creamos una segunda actividad *HelloActivity* con un `TextView` centrado en la pantalla y un texto de prueba desde el espacio de nombres **tools**:

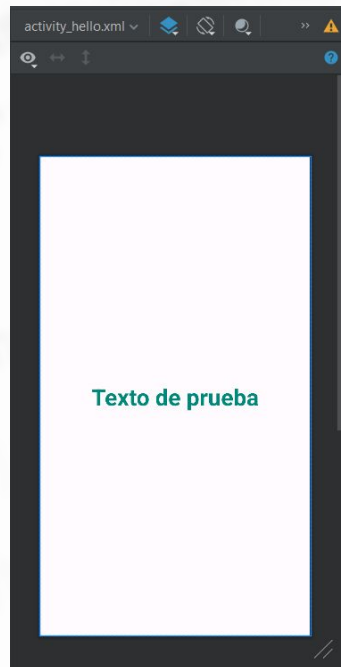
```
<FrameLayout...>
<TextView
    android:id="@+id/helloText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    tools:text="Texto de prueba"
    android:textSize="35sp"
    android:textStyle="bold"
    android:textColor="#00897B"
    android:layout_gravity="center"/>
```



1. INTERACCIONES

Y definimos la variable que almacena nuestro TextView pudiendo posteriormente añadirle el texto proveniente de *MainActivity*, introducido por el usuario:

```
class HelloActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_hello)  
  
        var textoHola = findViewById<TextView>(R.id.helloText)  
    }  
}
```



2. INTENTS

Un *Intent* es un objeto de mensajería que puedes usar para solicitar una acción de otro componente de una app.

Un objeto *Intent* tiene información que el sistema Android usa para determinar qué componente debe iniciar (como el nombre exacto del componente o la categoría que debe recibir el intent), además de información que el componente receptor usa para realizar la acción correctamente (por ejemplo, la acción que debe efectuar y los datos en los que debe actuar).

2. INTENTS

Ahora, en *MainActivity*, definimos un *Intent* en el que hay que declarar el contexto en el que actúa y la clase que inicia (en nuestro caso *HelloActivity*):

```
var btnSend = findViewById<Button>(R.id.buttonSend)
var userText = findViewById<EditText>(R.id.etName)

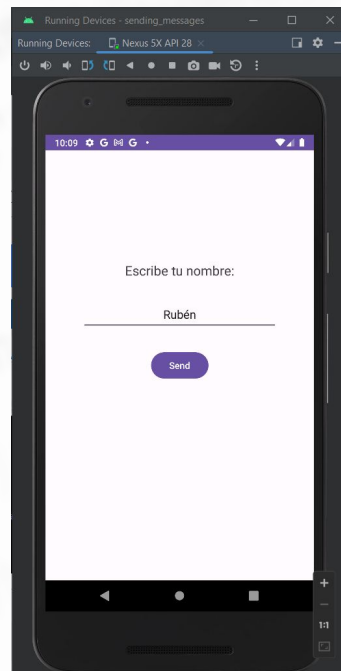
btnSend.setOnClickListener{
    var name = userText.text.toString()
    if (name.isNotEmpty()){
        var textIntent = Intent(this, HelloActivity::class.java)
        startActivity(textIntent)
    }
}
```



2. INTENTS

Aparte de iniciar clases, los Intents pueden llevarse otros elementos con ellos, como variables. Utilizamos entonces el método *putExtra* y le asignamos una clave (*extra_name*) al elemento que queremos enviar:

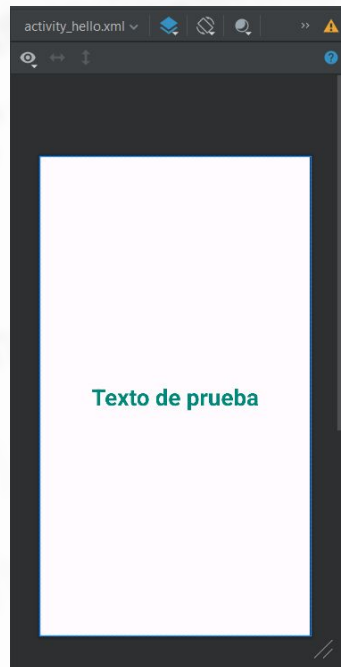
```
btnSend.setOnClickListener{
    var name = userText.text.toString()
    if (name.isNotEmpty()){
        var textIntent = Intent(this, HelloActivity::class.java)
        textIntent.putExtra("extra_name", name)
        startActivity(textIntent)
    }
}
```



2. INTENTS

Para recuperar los valores en *HelloActivity* basta con implementar el método *extras* llamando a la clave que le hemos asignado a la variable *name*:

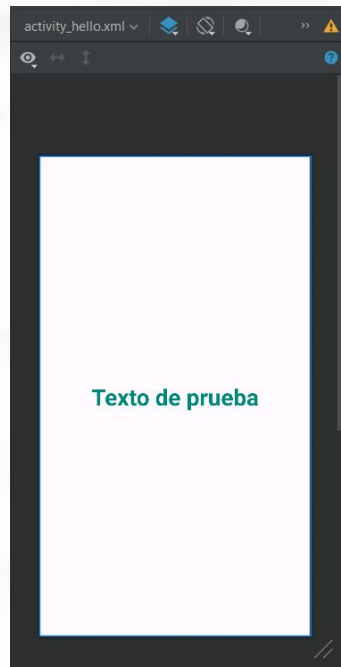
```
class HelloActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_hello)  
  
        var textoHola = findViewById<TextView>(R.id.helloText)  
  
        var nombre: String = intent.extras?.getString("extra_name").orEmpty()  
    }  
}
```



2. INTENTS

Con el método `var nombre = intent.extras["extra_name"]` se produce un error de nulabilidad porque Android no sabe si el *extra* existe o si tiene algo almacenado:

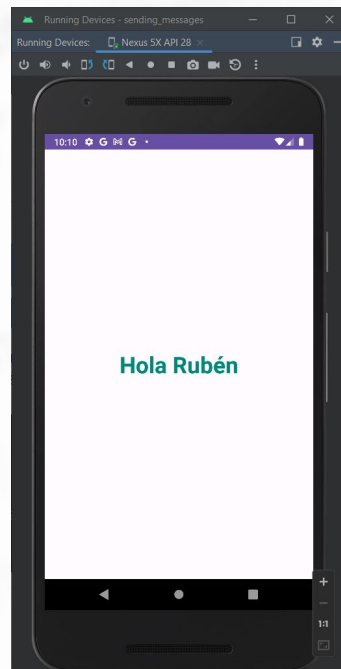
```
class HelloActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_hello)  
  
        var textoHola = findViewById<TextView>(R.id.helloText)  
  
        var nombre: String = intent.extras?.getString("extra_name").orEmpty()  
    }  
}
```



2. INTENTS

Ya solo nos queda asignarle al TextView el nombre introducido por el usuario y ejecutar la aplicación en el dispositivo virtual:

```
class HelloActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_hello)  
  
        var textoHola = findViewById<TextView>(R.id.helloText)  
        var nombre: String = intent.extras?.getString("extra_name").orEmpty()  
        textoHola.text = "Hola $nombre"  
    }  
}
```



3. PRIMERA APLICACIÓN

Realiza una aplicación de mensajería sencilla en la que la primera actividad muestre un EditText y un botón para enviar en la parte inferior.

La segunda pantalla será exactamente igual a la primera pero mostrará el mensaje recibido en la parte superior.

Si desde esta segunda pantalla se manda otro mensaje, la aplicación deberá volver a la primera pantalla y mostrar el mensaje enviado, también en la parte superior.

