Desarrollo de apps con Android

1. Introducción a Android

¿Qué es Android?

- Sistema operativo para móviles basado en Kernel Linux.
- Interfaz de usuario para pantalla táctiles.
- El más utilizado en smartphones a nivel nacional.
- Explota el potencial de dispositivos como relojes, TVs, gafas y coches.
- 2.662.566 apps en Google Play (18 de septiembre 2024).
- Open source.
- Forma parte del consorcio Handset Alliance creado en 2007: desarrollo de estándares abiertos para móviles a través del diseño y difusión de Android.

Android: interacción de usuario

- Modo táctil: deslizar, pulsar y apretar.
- Teclados virtuales para caracteres, números y emojis.
- Ofrece soporte para Bluetooth, controladores USB y periféricos.

Android: sensores

Dispone de sensores que detectan y reaccionan antes las acciones del usuario:

- Los contenidos del dispositivo rotan según se necesite.
- Según vamos caminando se va ajustando la posición en el mapa.
- Permite controlar coches y juguetes.

Android: pantalla de inicio

- Iconos para ejecución de apps.
- *Widgets* de contenido actualizado.
- Puede tener varias páginas.
- Organización de apps por carpetas.
- Asistente de voz.

Android: ejemplos de apps

RTVE Audio

Pokémon GO

Messenger

Android: Software Developer Kit (SDK)

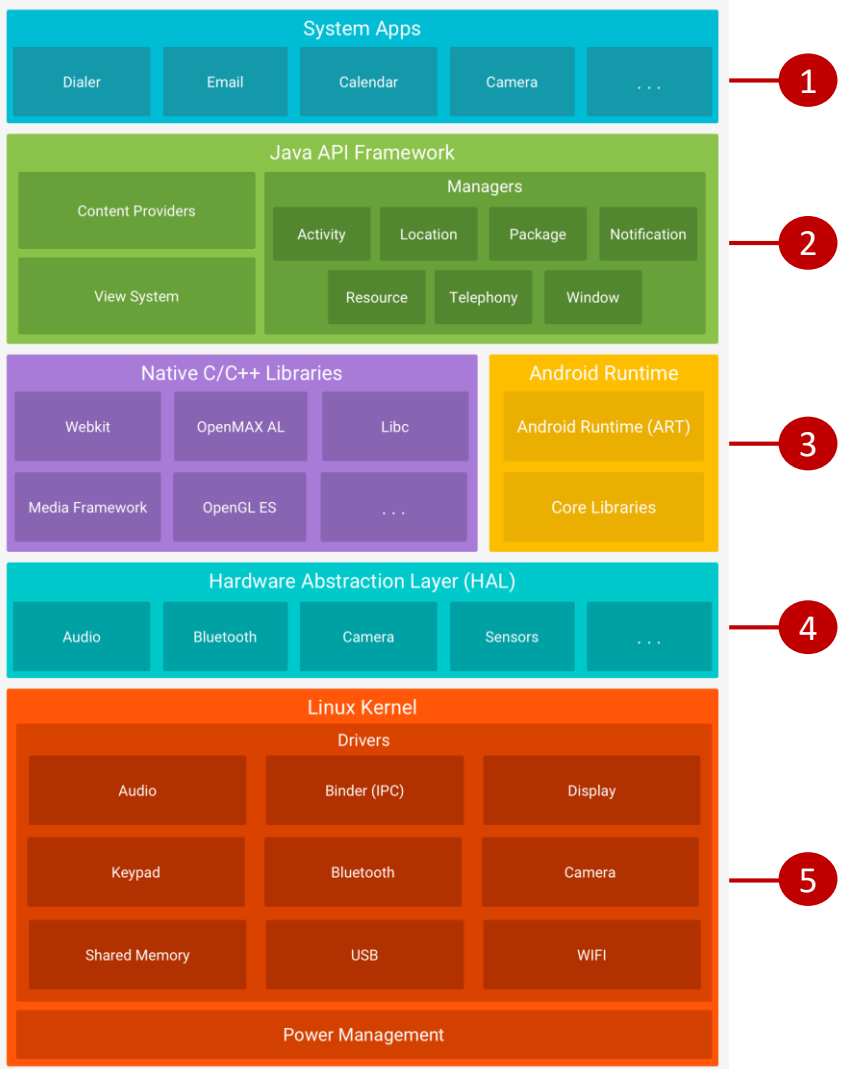
- Herramientas de desarrollo (depuración, monitorización, edición).
- Librerías (mapas, dispositivos *wearables*).
- Dispositivos virtuales (emuladores).

1.1. Arquitectura de la plataforma Android

Android stack

Componentes principales:

1. Apps del sistema.
2. Android OS API en Java framework.
3. Exponer APIs nativas; ejecutar apps.
4. Exponer capacidades de los dispositivos HW.
5. Núcleo del sistema operativo.



Apps del sistema

- Las apps del sistema no tienen un estado especial entre las apps que el usuario elige instalar (excepción, la configuración del sistema).
- Las apps del sistema proporcionan capacidades/funcionalidades clave para los desarrolladores de apps.
- Ejemplo: Se puede implementar una app de usuario que haga uso de una app del sistema para enviar un SMS.

Java API Framework

Todos los elementos del sistema operativo Android están disponibles a través de APIs escritas en Java.

- Jerarquía de clases en vistas *(*`View`*)* para crear pantallas de interfaz de usuario.
- Un gestor de recursos como por ejemplo los strings para los distintos idiomas, imágenes, ficheros de layout, etc.
- Gestor de notificaciones.
- Gestor de actividades para el ciclo de vida y la navegación.
- Proveedor de contenidos para acceder a los datos de otras apps, como la app de contactos.

Android runtime

Cada app se ejecuta en su propio proceso con su propia instancia de Android Runtime (ART).

Librerías C/C++

- Muchos componentes del sistema Android, como por ejemplo, ART y HAL, están implementados a través de código nativo que requiere librerías de C/C++.
- La plataforma Android proporciona funcionalidad de algunas de esas librerías nativas a las apps.
- Si se desarrolla una app que requiere C o C++, se puede hacer uso del NDK (*Native Development Kit*) de Android. De esta manera, se puede acceder a algunas de esas librerías de plataformas nativas directamente desde el código de la propia app.

Hardware Abstraction Layer (HAL)

- Interfaces estándares que dan acceso a capacidades de dispositivos hardware. Ejemplos: Cámara, Audio y Bluetooth.

Kernel Linux

- Gestión de los hilos y de memoria a bajo nivel.
- Elementos de seguridad.
- Drivers (Audio, Cámara, Bluetooth, USB, WIFI…).

Evolución de Android (1/3)

- Lanzamiento de Android en 2008

Codename	Versión	Release	Nivel de API
Honeycomb	3.0 - 3.2.6	Feb 2011	11 – 13
Ice Cream Sandwich	4.0 - 4.0.4	Oct 2011	14 – 15
Jelly Bean	4.1 - 4.3.1	Julio 2012	16 – 18
KitKat	4.4 - 4.4.4	Oct 2013	19 – 20
Lollipop	5.0 - 5.1.1	Nov 2014	21 – 22

T-Mobile G1 / HTC Dream

Historia de Android y Versiones para más detalle.

Evolución de Android (2/3)

Codename	Versión	Release	Nivel de API
Marshmallow	6.0 - 6.0.1	Oct 2015	23
Nougat	7.0 - 7.1	Sept 2016	24 - 25
Oreo	8.0 - 8.1	Sept 2017	26 - 27
Pie	9.0	Agosto 2018	28
Android 10	10.0	Sept 2019	29
Android 11	11.0	Sept 2020	30
Android 12	12.0	Oct 2021	31, 32

Evolución de Android (3/3)

Codename	Versión	Release	Nivel de API
Android 13	13.0	Agosto 2022	33
Android 14	14.0	Octubre 2023	34
Android 15 (beta)	-	-	

Posibilidad de participación en las versiones beta:

https://www.google.com/android/beta

1.2. Desarrollo de apps con Android

¿Qué es una app de Android?

- Una o más pantallas interactivas.
- Escrita en Java o Kotlin (y XML).
- Utiliza el Software Development Kit (SDK) de Android.
- Utiliza librerías Android y el Framework para aplicaciones de Android.
- Se ejecuta a través de la máquina virtual de Android Runtime (ART).

Bloques de construcción de una app en Android

- **Recursos:** layout (interfaz de usuario), imágenes, strings y colores a través de ficheros XML y ficheros multimedia.
- **Componentes:** actividades y servicios.
- **Manifiesto:** información sobre la app de cara a su ejecución *(runtime*).
- **Configuración para su construcción *(build*):** versiones APK en los ficheros de configuración de Gradle.

Google Play

Las apps se publican a través de Google Play:

- Tienda oficial de apps para Android.
- Servicio de distribución digital operado por Google.

Google Play

Desafíos en el desarrollo de apps con Android

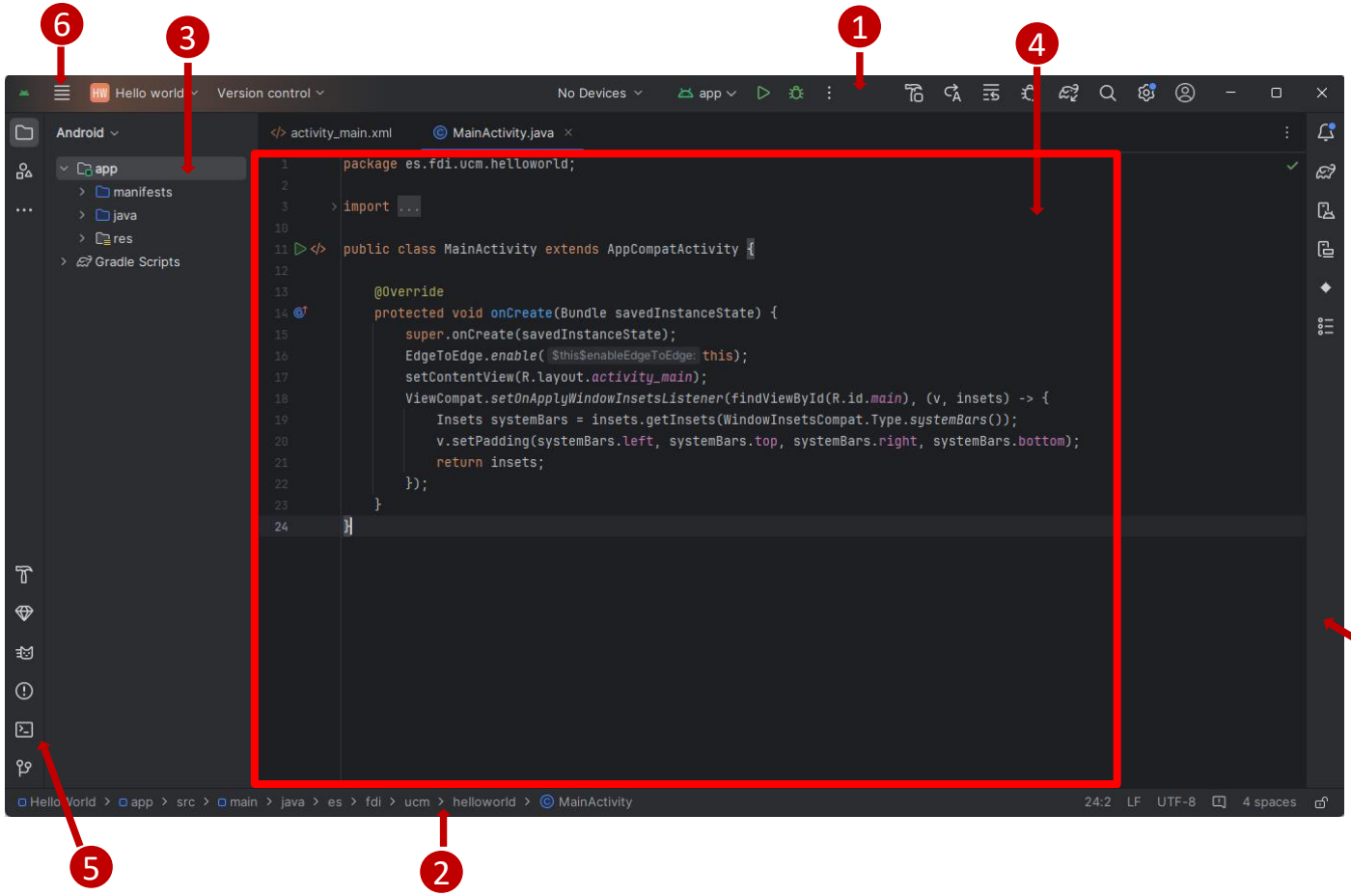
- Varios tamaños de pantalla y resoluciones.
- Rendimiento: permite apps responsivas y fluidas.
- Seguridad en el código fuente y en los datos de los usuarios.
- Compatibilidad entre las diferentes versiones de Android.
- Marketing: entender el mercado y los potenciales usuarios de la app objetivo.

1.3. Android Studio

¿Qué es Android Studio?

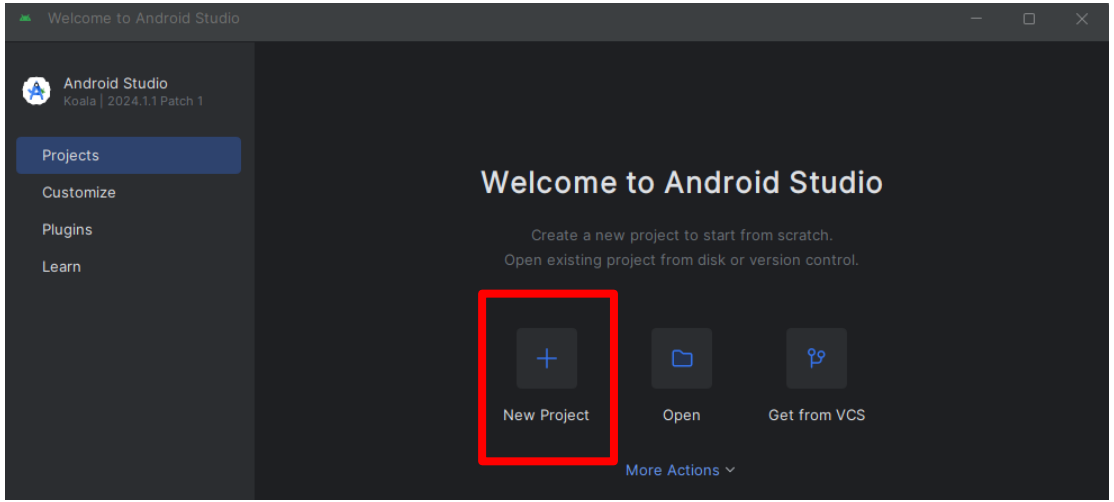
- IDE oficial para Android.
- Plantillas de creación de proyectos.
- Editor de layout (diseño de la interfaz de usuario).
- Herramientas para pruebas.
- Construcción (*build*) basada en Gradle.
- Depuración y consola de logs.
- Emuladores.

Interfaz de Android Studio

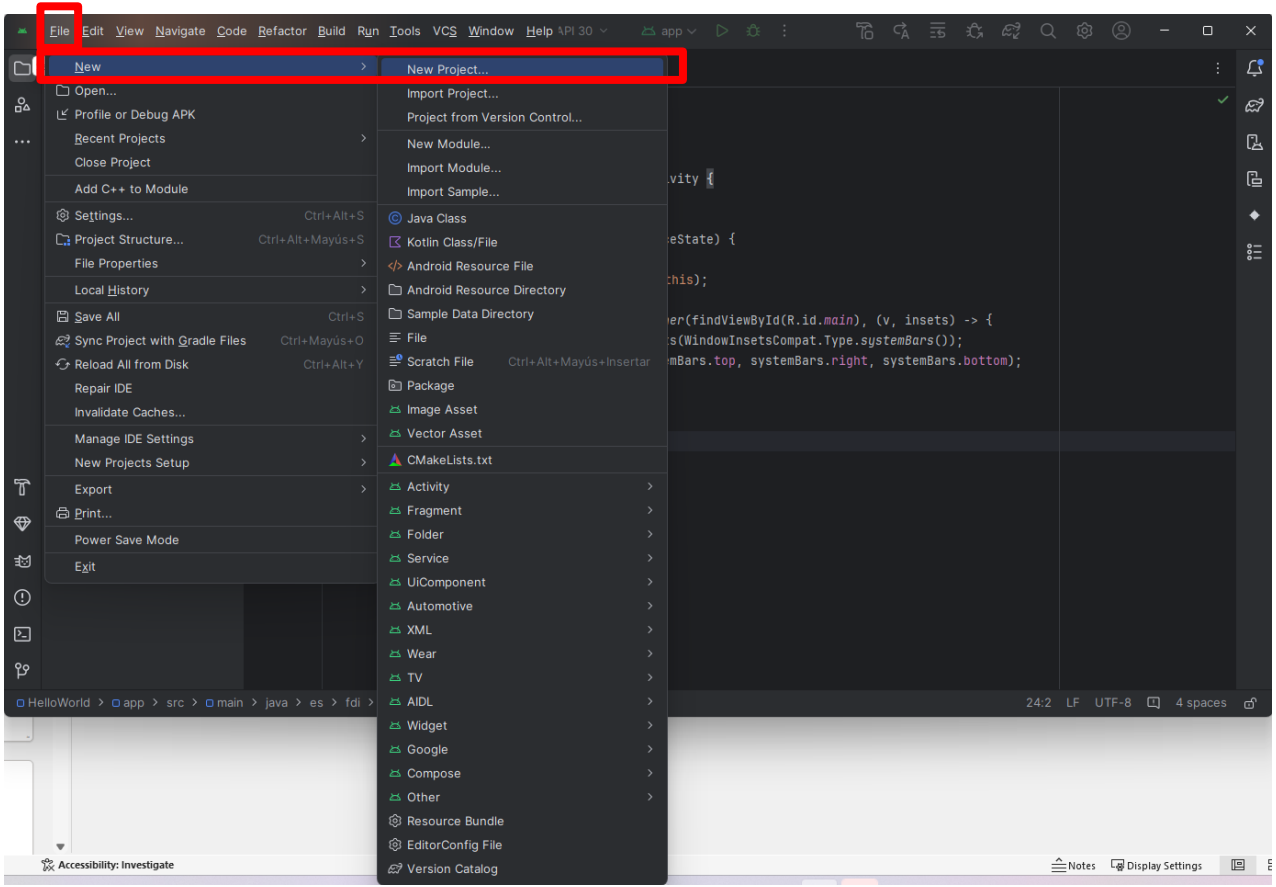


1. Barra herramientas
2. Barra navegación
3. Panel del proyecto
4. Editor
5. Pestañas para otros paneles
6. Menú

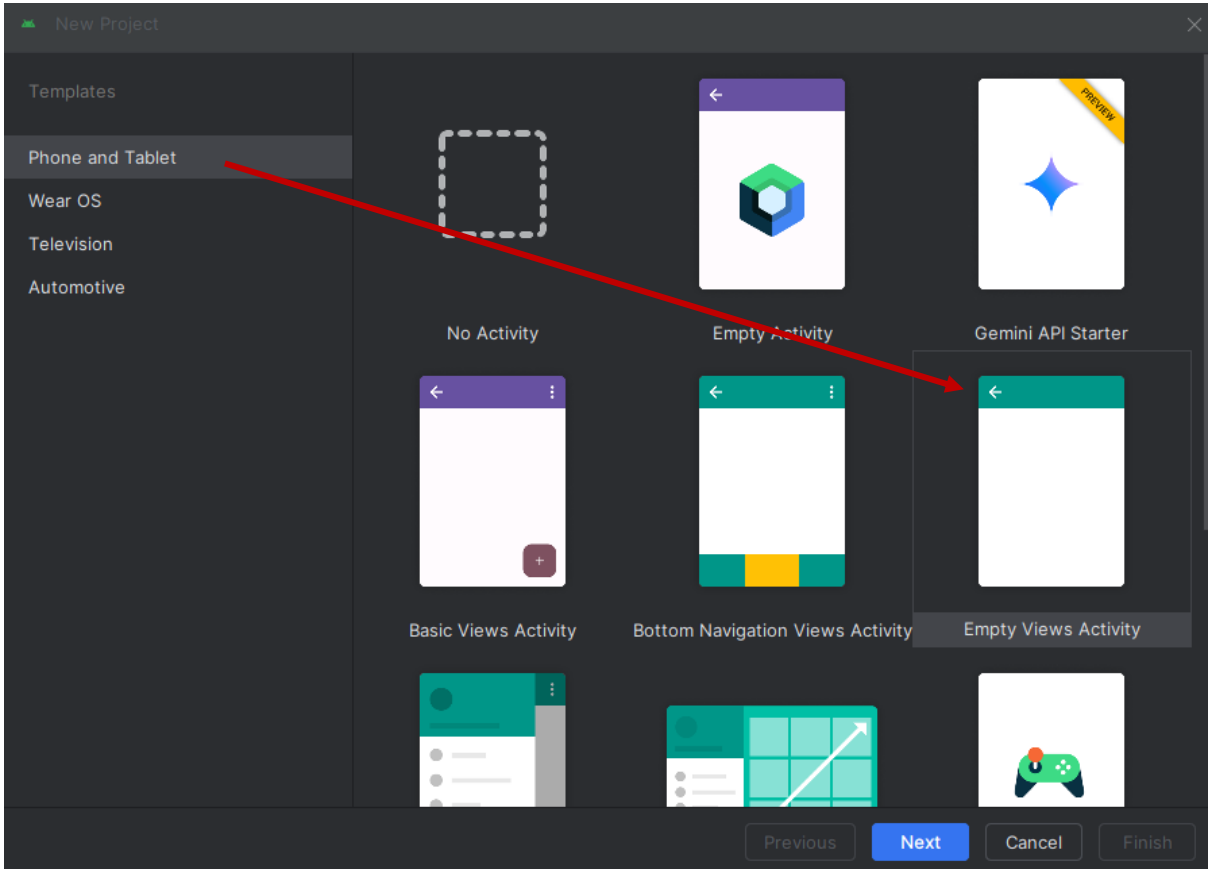
Android Studio: Creación de un proyecto (1/2)



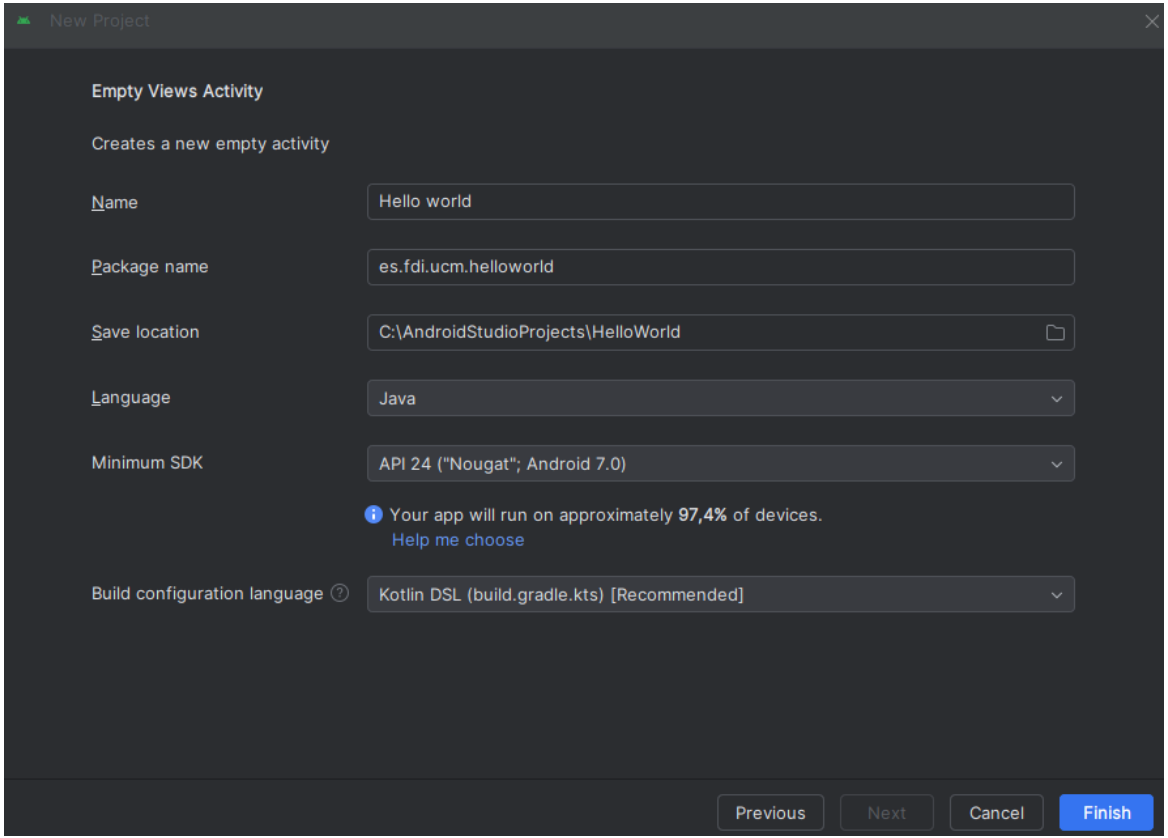
Android Studio: Creación de un proyecto (2/2)



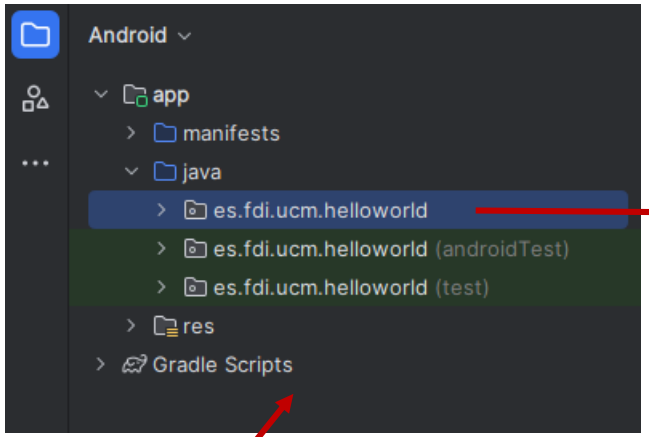
Selección de la plantilla para la actividad principal



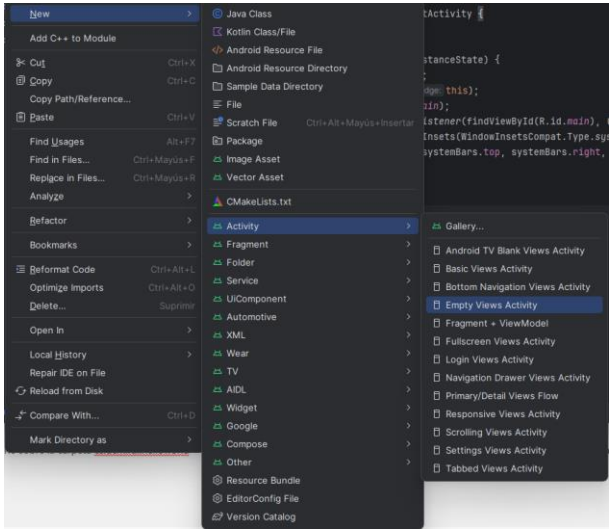
Configuración de la app

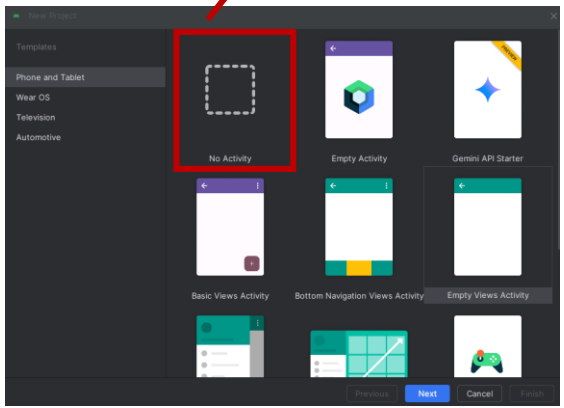


Creación de la actividad principal (1/4)

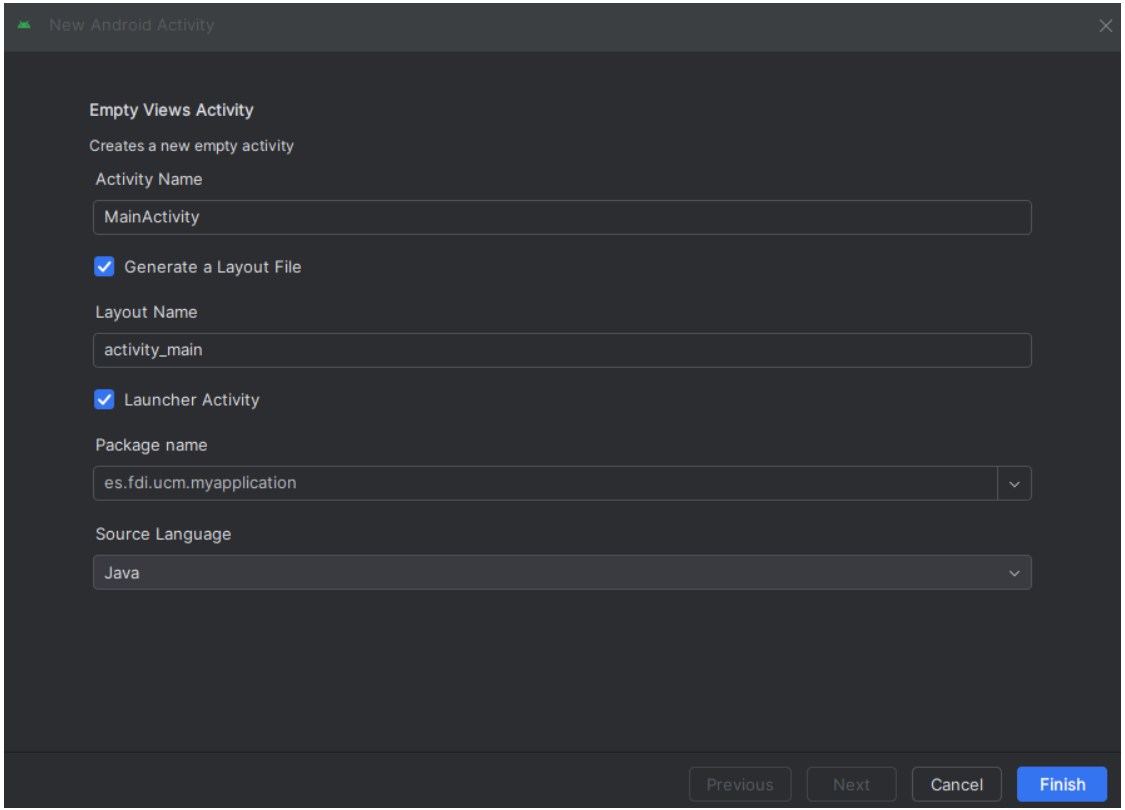


New > Activity > Empty Views Activity

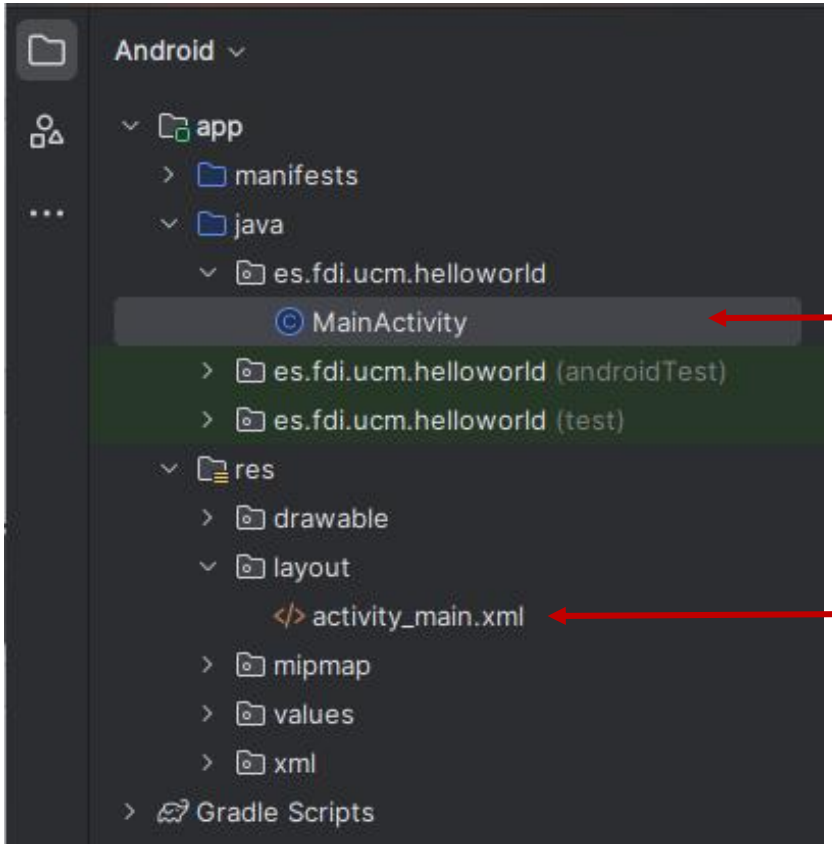




Creación de la actividad principal (2/4)



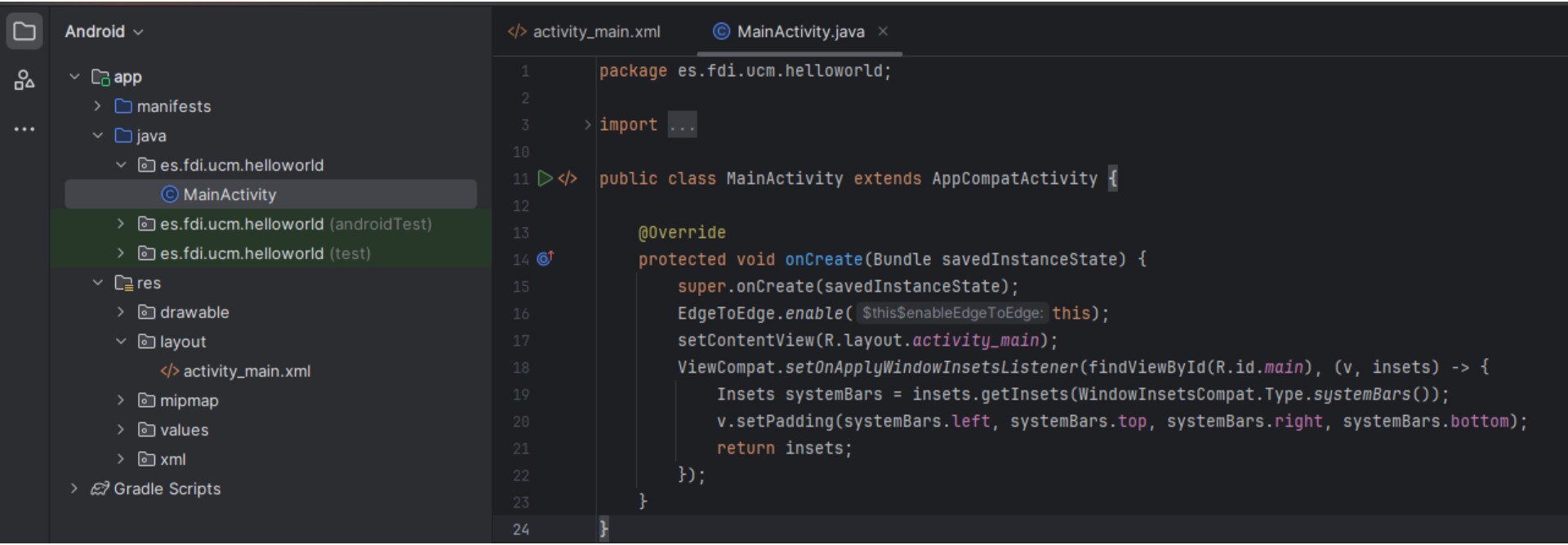
Creación de la actividad principal (3/4)



Actividad principal

Layout de la actividad principal

Creación de la actividad principal (4/4)

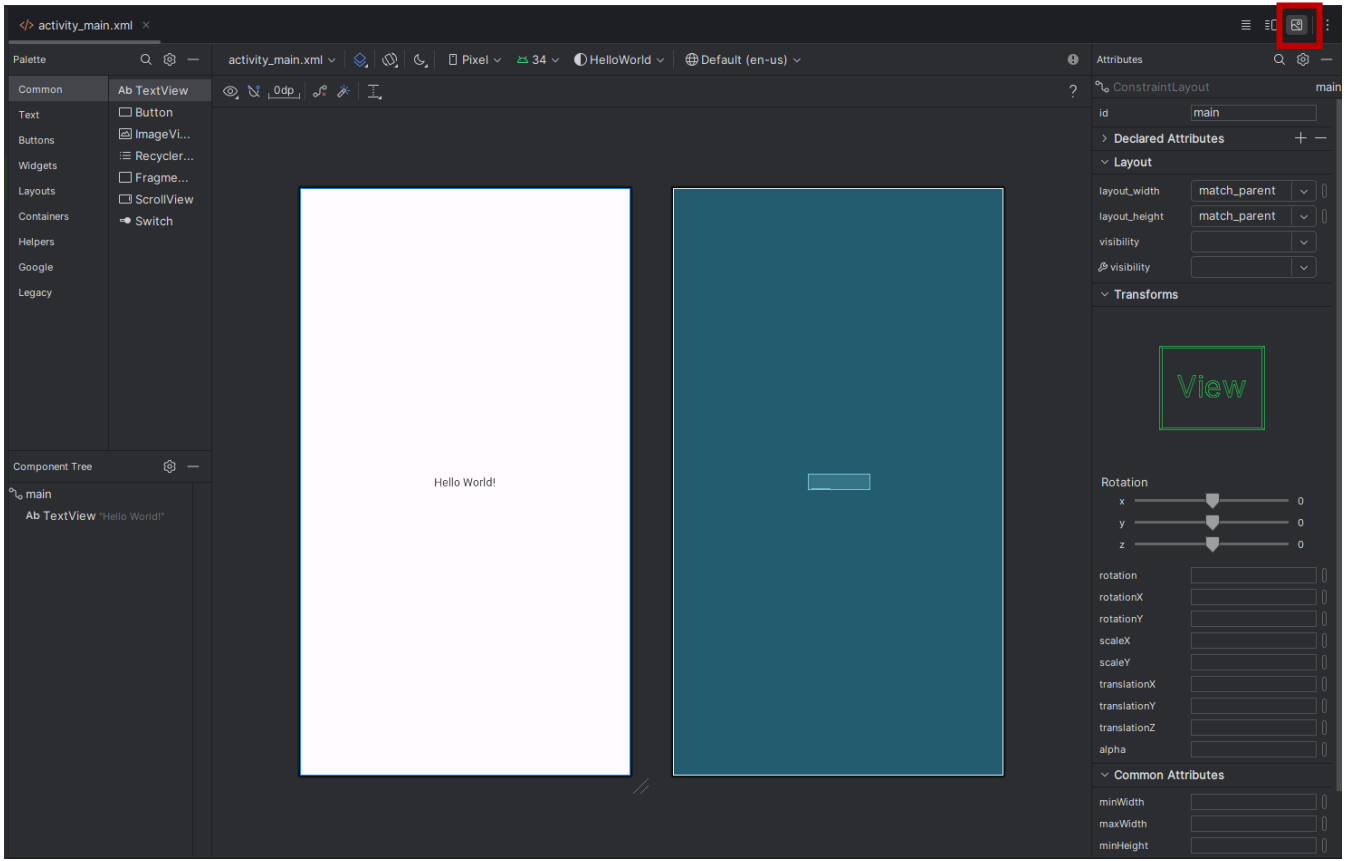


MainActivity.java

MainActivity.java

```java
package es.fdi.ucm.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_main);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }
}
```

activity_main.xml (diseño)

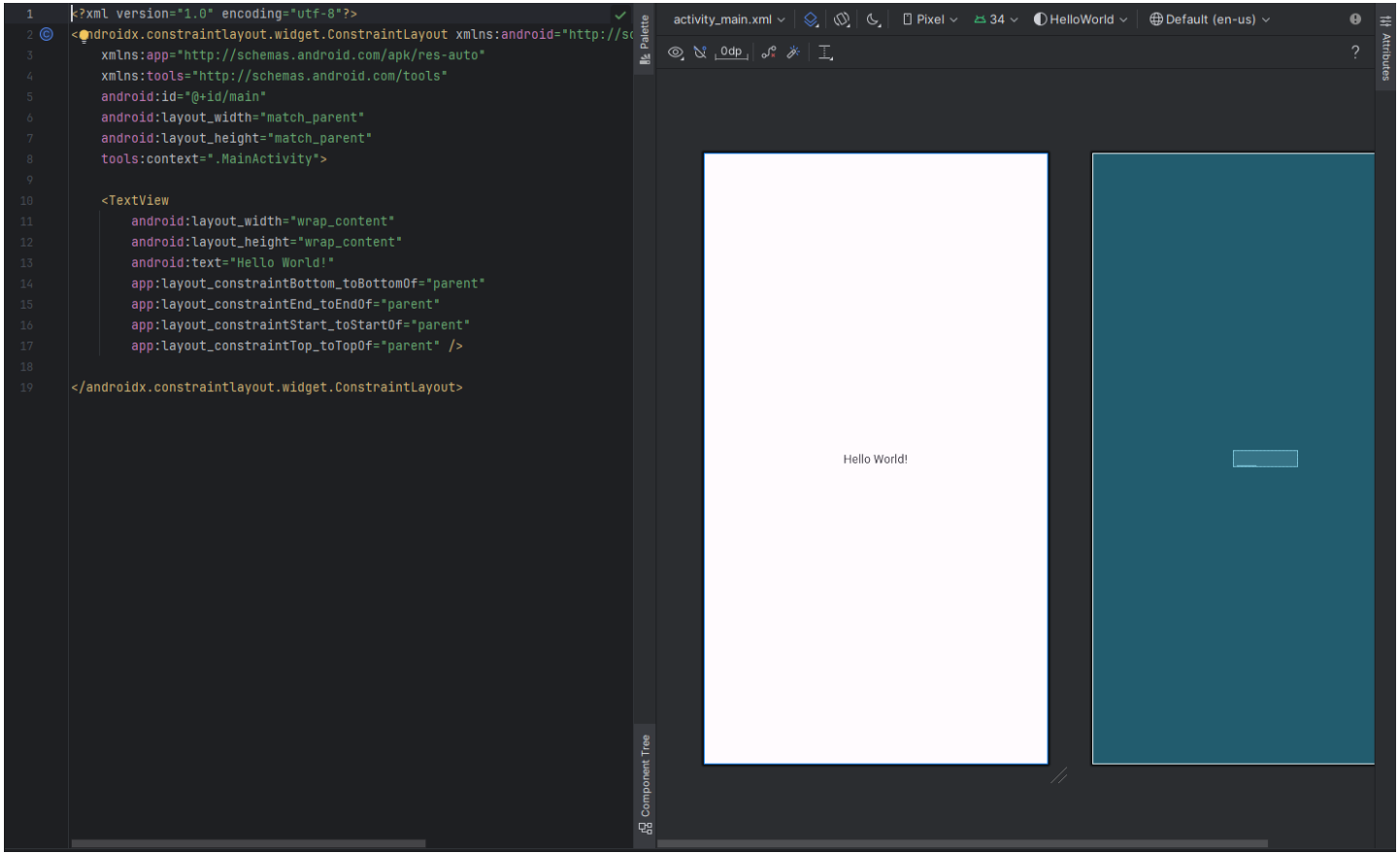


activity_main.xml (código)

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout 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:id="@+id/main"
    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" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_main.xml (diseño+código)



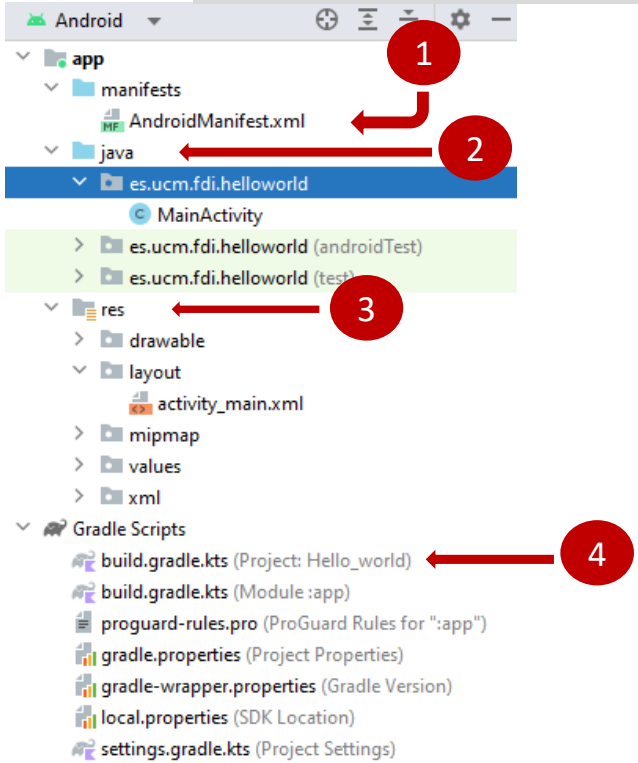

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://sc
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    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" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Seiri (ordenación).
Seiton (sistematización).
Seiso (limpieza).
Seiketsu (estandarización).
Shutsuke (disciplina).

5S

Estructura del proyecto

1. **manifests –** Fichero *AndroidManifest.xml:* descripción de la app de cara a su ejecución con Android runtime.
2. **java –** Paquetes del código fuente de Java.
3. **res –** Recursos (XML): layout, strings, imágenes, dimensiones, colores...
4. **build.gradle—**Ficheros build de Gradle.

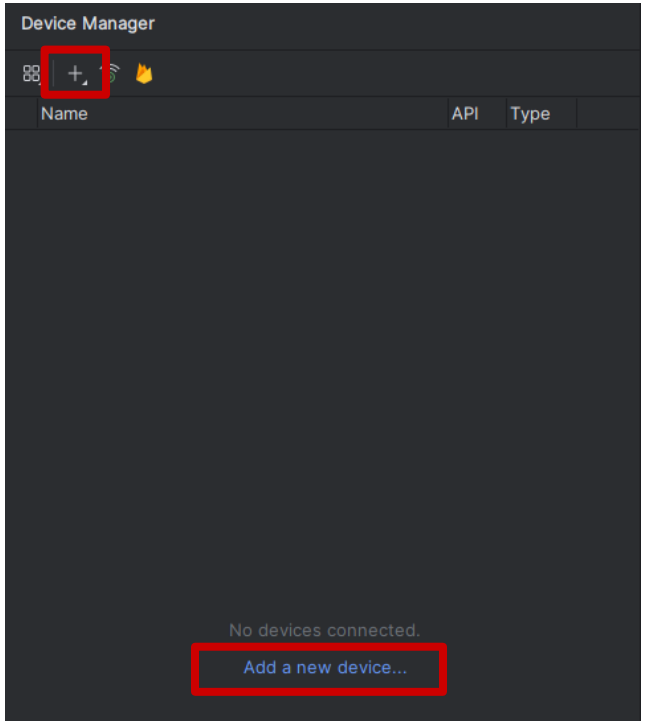


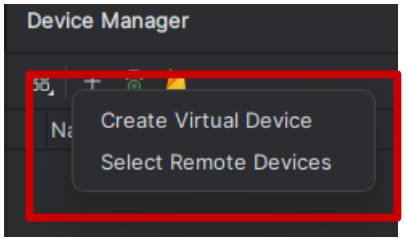
Sistema build de Gradle

- Subsistema construido en Android Studio.
- Tres build.gradle:
  - project
  - module
  - settings
- local.properties
- No es necesario conocer Gradle a bajo nivel.
- Más detalle en: https://gradle.org/

Creación de un dispositivo virtual

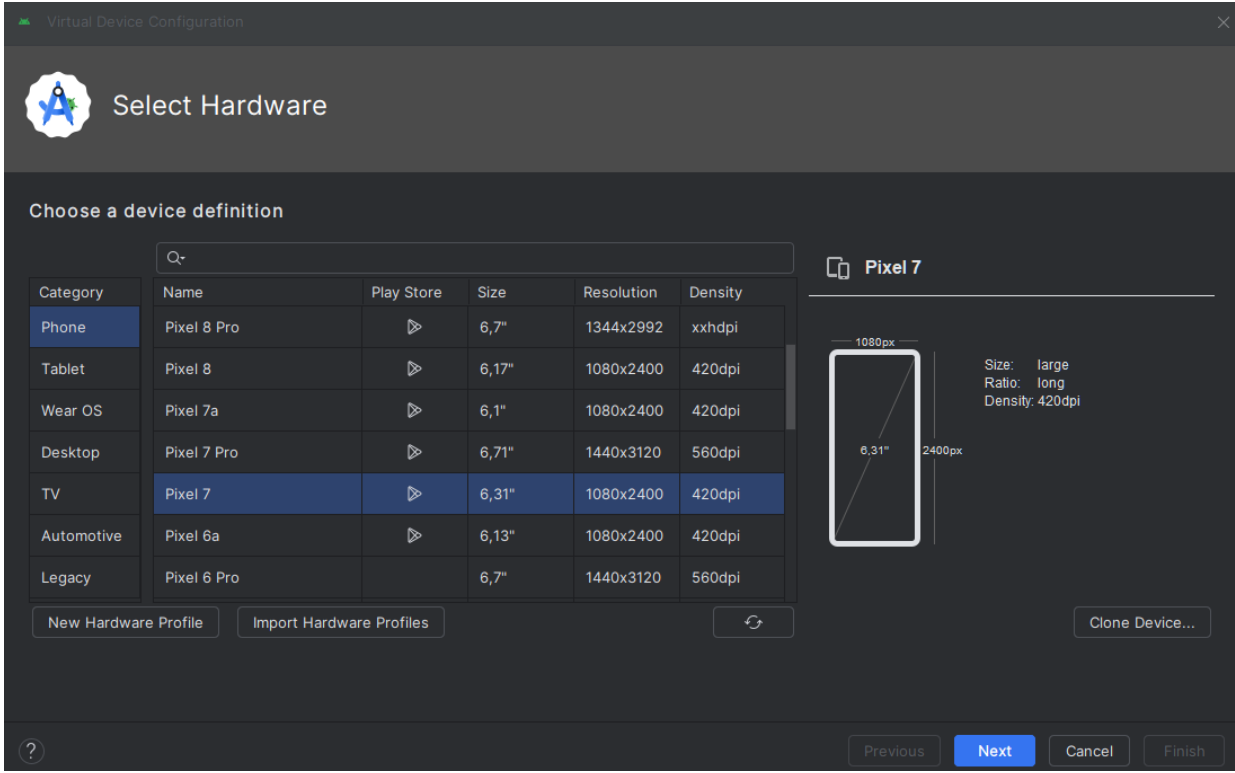
Utilizar emuladores para probar la app en diferentes versiones de Android y dispositivos: **Tools > Device Manager**





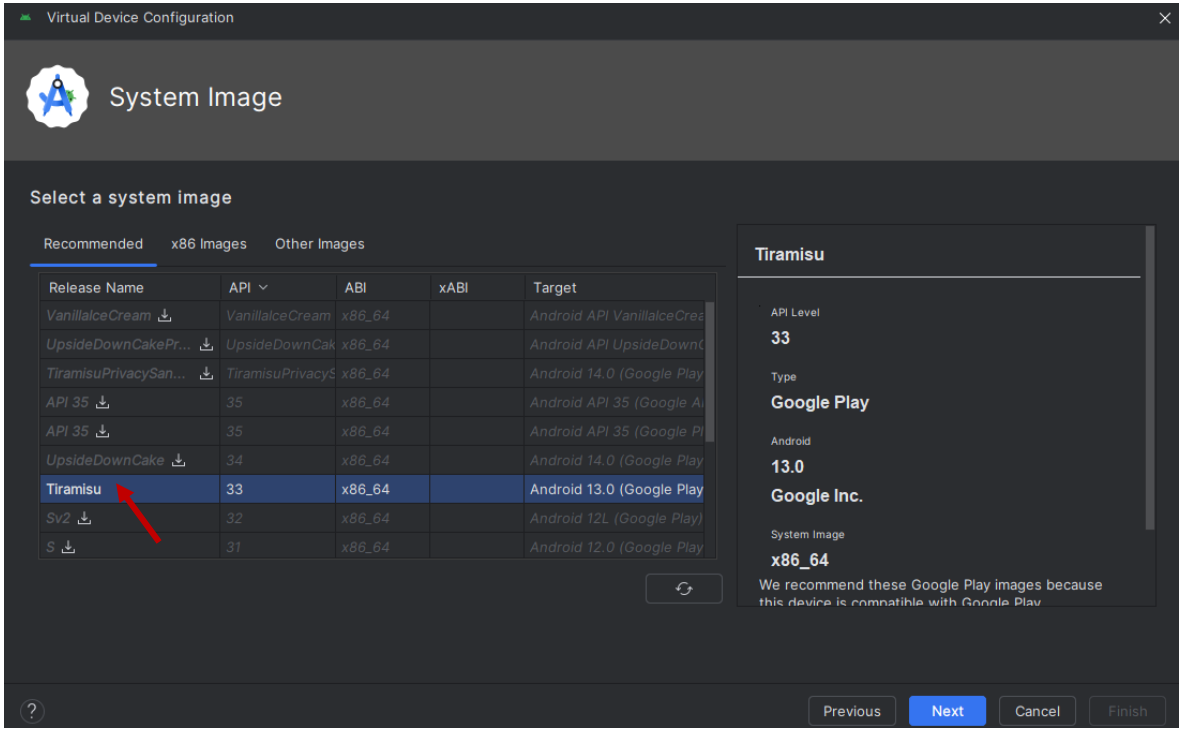
Configuración de un dispositivo virtual (1/3)

1. Seleccionar hardware.



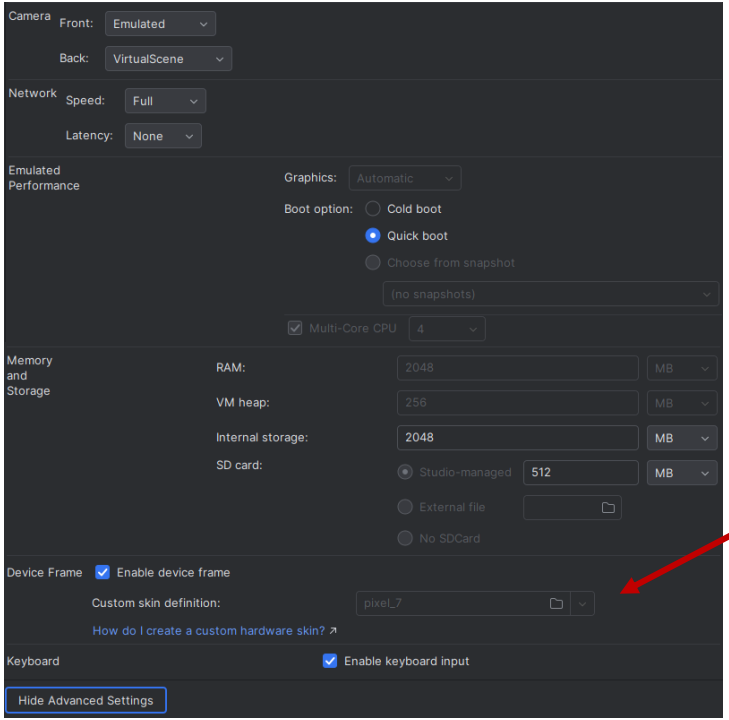
Configuración de un dispositivo virtual (2/3)

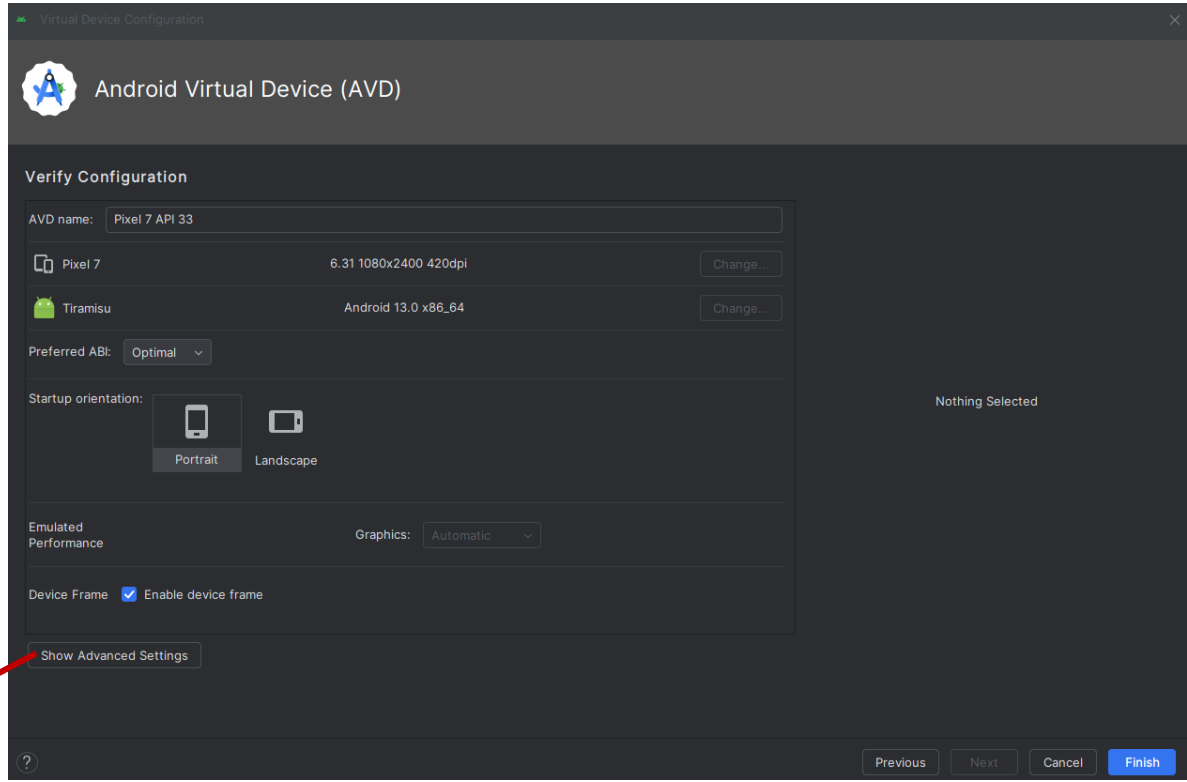
2. Seleccionar versión de Android: seleccionar imagen del sistema (descargar primero para instalar si no está en Android Studio).



Configuración de un dispositivo virtual (3/3)

3. Realizar ajustes finales y terminar.



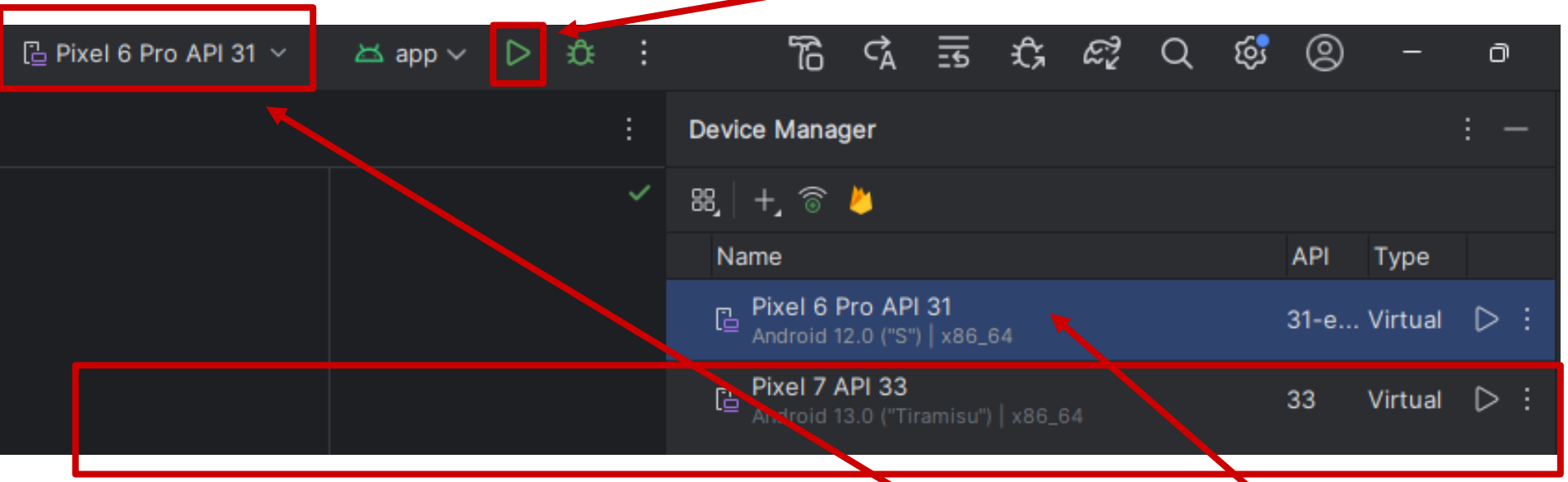


Dispositivos virtuales: incidencias

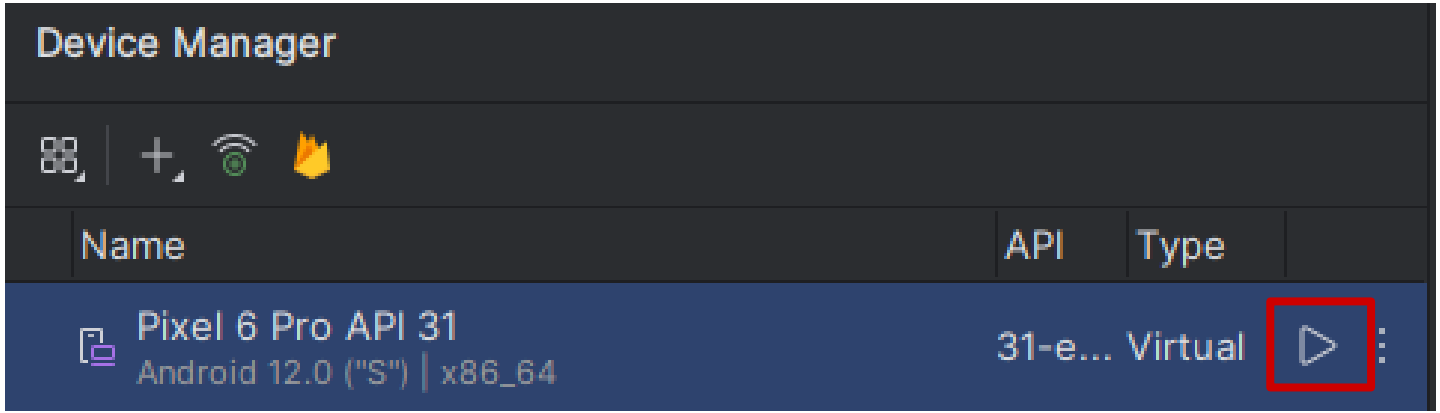
- Problemas conocidos para el emulador de Android:

https://developer.android.com/studio/run/emulator-troubleshooting?hl=es-419

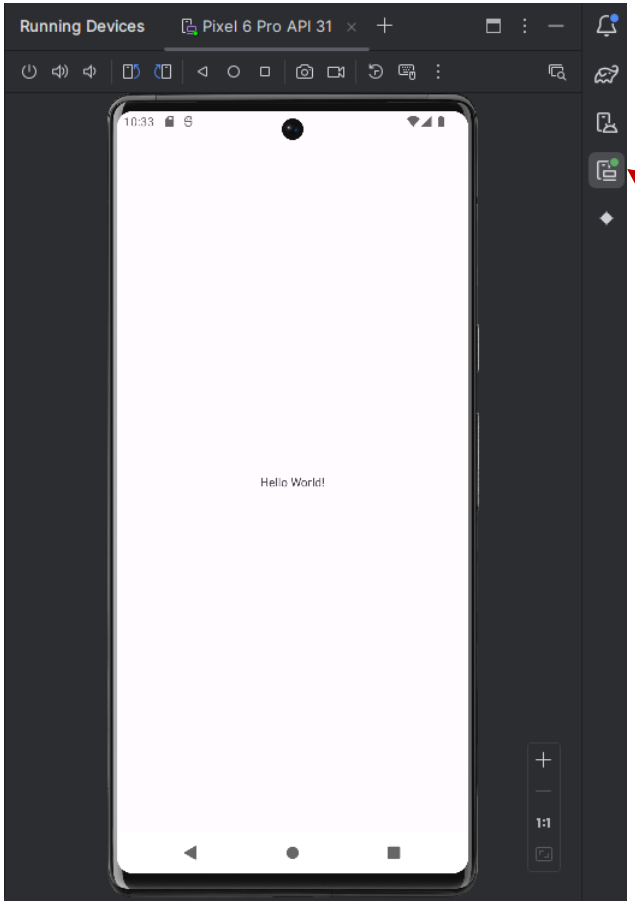
Ejecución de una app (1/4)



Ejecución de una app (2/4)

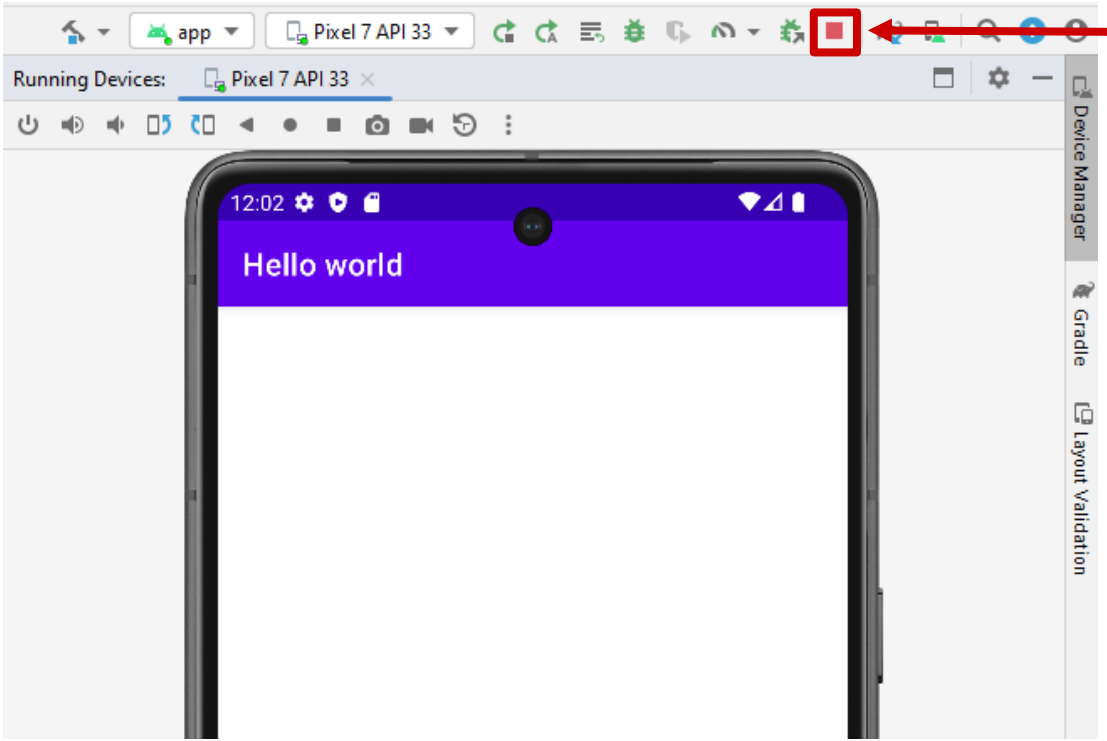


Ejecución de una app (3/4)



Pestaña
Running Devices

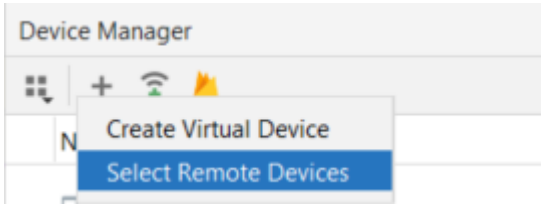
Ejecución de una app (4/4)

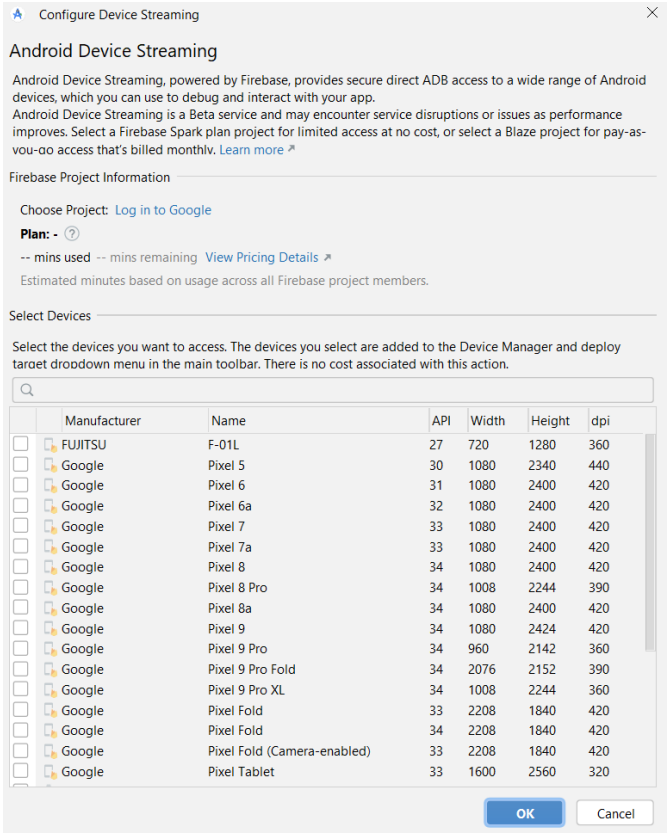


3. Parar

Android Device Streaming

1. Android Device Streaming permite ejecutar dispositivos en la nube que dispone Firebase.
2. Device Manager: **Select Remote Devices**





Ejecución de una app desde un dispositivo físico

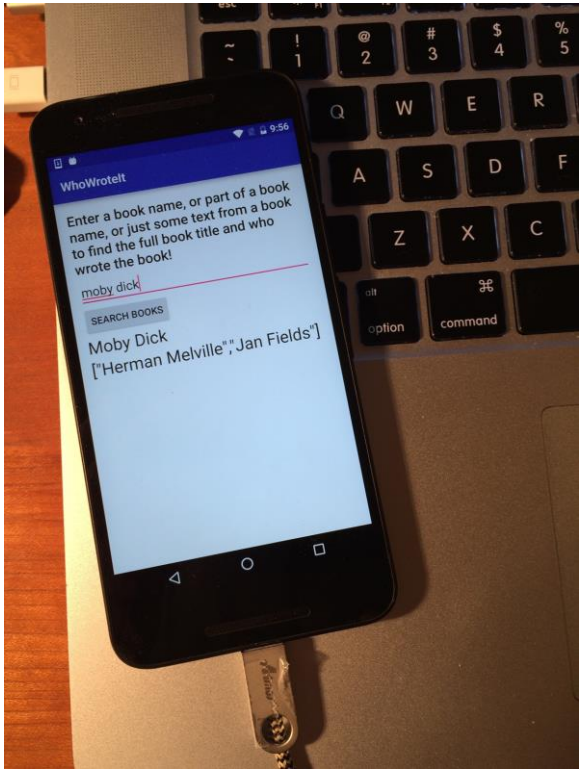
1. Activar las opciones del Desarrollador:
    a. **Settings (Ajustes) > About phone (Acerca del teléfono)**
    b. **Software Info (Información de software) >** Pulsar siete veces **Build number (Número de compilación)**
2. Activar depuración USB:
    a. **Settings (Ajustes) > Developer Options (Opciones de desarrollador) > USB Debugging (Depuración por USB)**
3. Conectar el teléfono al ordenador a través de un cable.
4. En Android Studio: **Tools > SDK Manager > Android SDK > SDK Tools: Marcar opción Google USB Driver**.

Windows/Linux configuraciones adicionales:

- Utilizar dispositivos hardware

Drivers de Windows:

- Drivers OEM USB



Pair Devices using Wi-Fi

- Para Android 11 o superior.
- Posibilidad de acceder a dispositivos a través de Wi-Fi habilitando las opciones del Desarrollador y la depuración inhalámbrica en el dispositivo.

1.4. La interfaz de usuario

View

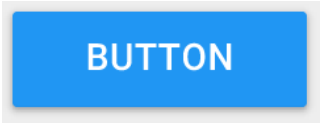
¿Qué es una View?

`View`: bloque básico de Android utilizado para la implementación de la interfaz de usuario en un móvil. Ejemplos de subclases de `View`:

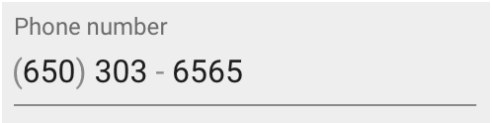
- Mostrar texto (TextView)
- Diferentes cajas de texto (EditText: Plain Text, Password, E-mail…)
- Botones (Button, ImageButton…)
- Menús (Menu)
- Desplazamiento (ScrollView, RecyclerView)
- Imágenes (ImageView)

Ejemplos de View

Button

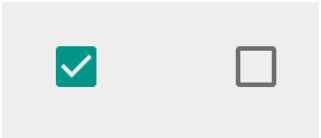


Phone



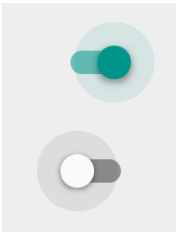
ImageView

CheckBox



RadioButton

Switch



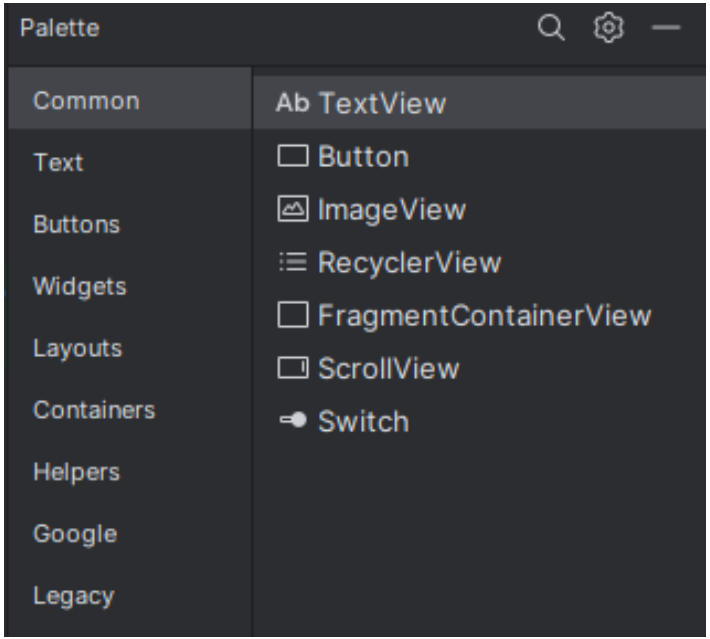
View: atributos

- Color, dimensiones, posicionamiento.
- Puede tener el foco (ej., seleccionado para recibir la entrada del usuario).
- Puede ser interactivo (responder a clics del usuario).
- Puede estar visible o no.
- Puede estar relacionado con otras *views*.

View: creación

- Android Studio layout editor: representación visual de XML.
- Editor XML editor.
- Código Java.

View: creación desde el editor Android Studio Layout



View: creación desde fichero XML (1/2)

```
<TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/myBackgroundColor"
        android:text="@string/count_initial_value"
        android:textColor="@color/colorPrimary"
        android:textSize="@dimen/count_text_size"
        android:textStyle="bold"
/>
```

View: creación desde fichero XML (2/2)

- Atributos:

android:<nombre_propiedad>="<valor_propiedad>"

Ejemplo: android:layout_width="match_parent"

android:<nombre_propiedad>="@<tipo_recurso>/recurso_id"

Ejemplo: android:text="@string/button_label_next"

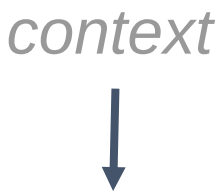
Para el identificador único de la `View`:

android:id="@+id/<valor_view_id>"

Ejemplo: android:id="@+id/show_count"

View: creación con código Java

En una `Activity`:




```
TextView myText = new TextView(this);
myText.setText("¡Hola, mundo!");
```

TextView

TextView

- `TextView` subclase de `View` para texto en una línea o multi-línea.
- Controlado por atributos del layout.
- Reconoce etiquetas HTML.
- Asignar texto:
  - De manera estática con fichero de recursos **strings.xml**.
  - De manera dinámica por código o por otra fuente.

TextView: creación a partir de XML

```
<TextView android:id="@+id/textview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/my_story"/>
```

TextView: atributos comunes

`android:text`—texto a mostrar

`android:textColor`—color del texto

`android:textAppearance`—estilo o tema predefinido

`android:textSize`—tamaño del texto en `sp`

`android:textStyle`—`normal`, `bold`, `italic`, o `bold|italic`

`android:typeface`—`normal`, `sans`, `serif`, o `monospace`

`android:lineSpacingExtra`—espacio extra entre líneas en `sp`

TextView: formatear enlaces web

En fichero **activity_main.xml**:

```
<TextView
    android:id="@+id/article"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:autoLink="web"
    android:text="https://www.ucm.es"/>
```

TextView: creación con código Java

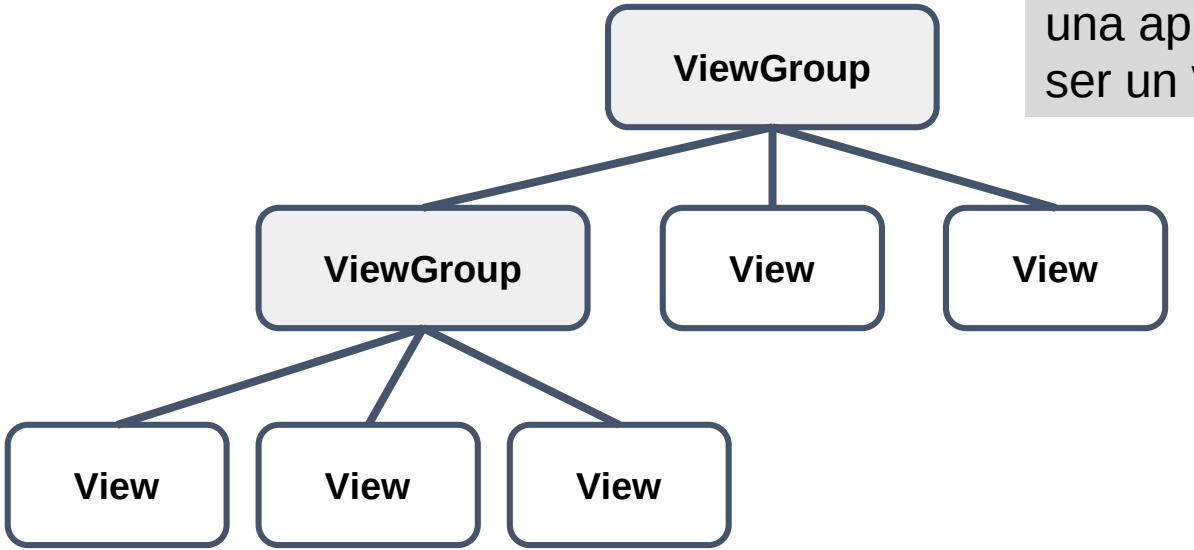
```
TextView myTextview = new TextView(this);
myTextView.setWidth(LayoutParams.MATCH_PARENT);
myTextView.setHeight(LayoutParams.WRAP_CONTENT);
myTextView.setMinLines(3);
myTextView.setText(R.string.my_story);
myTextView.append(userComment);
```

ViewGroup

ViewGroup

- `ViewGroup` es una subclase de `View` que contiene elementos `View`.
- Ejemplos:
  - `ScrollView`: Contiene un elemento y permite desplazamiento *(scrolling)*.
  - `ConstraintLayout`: Posiciona elementos de la interfaz de usuario haciendo uso de conexiones de restricción a otros elementos y a los límites del layout (diseño).
  - `RecyclerView`: Contiene una lista de elementos y permite desplazamiento a través de elementos que se añaden y eliminan de manera dinámica.

Jerarquía de elementos View



La `View` raíz que define una IU en una app de Android siempre va a ser un `ViewGroup`.

Buenas prácticas para jerarquías de View

- La disposición de la jerarquía de elementos de tipo *View* puede afectar al rendimiento de la app.
- Intentar utilizar el número mínimo posible.
- Procurar mantener la jerarquía plana, es decir, limitar el número de elementos de tipo **View** y *ViewGroup* anidados.

ScrollView

¿Cómo mostrar textos muy largos?

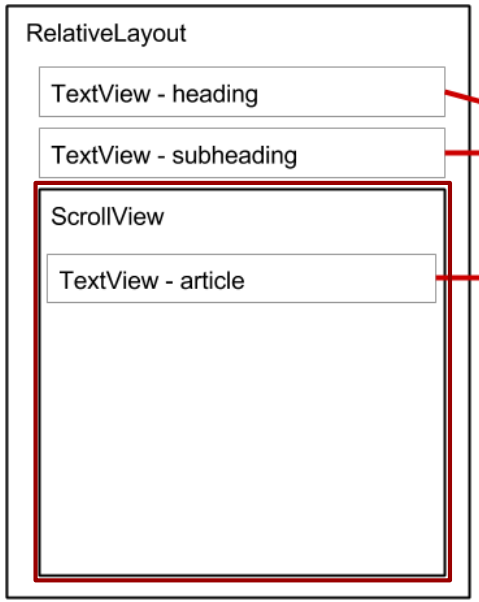
- Noticias, artículos, etc.
- Para añadir un barra de desplazamiento (*scroll*) a un `TextView`, integrarlo dentro de `ScrollView`.
- Para considerar varios elementos, utilizar un *`ViewGroup`* (ej., `LinearLayout`) en conjunción con *`ScrollView`*.

ScrollView

- `ScrollView` es subclase de `FrameLayout`.
- Mantiene todo el contenido en memoria.
- No va muy bien para textos muy largos o layouts complejos.
- No anidar *views* que tengan barras de desplazamiento.
- Utilizar `HorizontalScrollView` para barras de desplazamiento horizontales.
- Utilizar `RecyclerView` para mostrar listas.

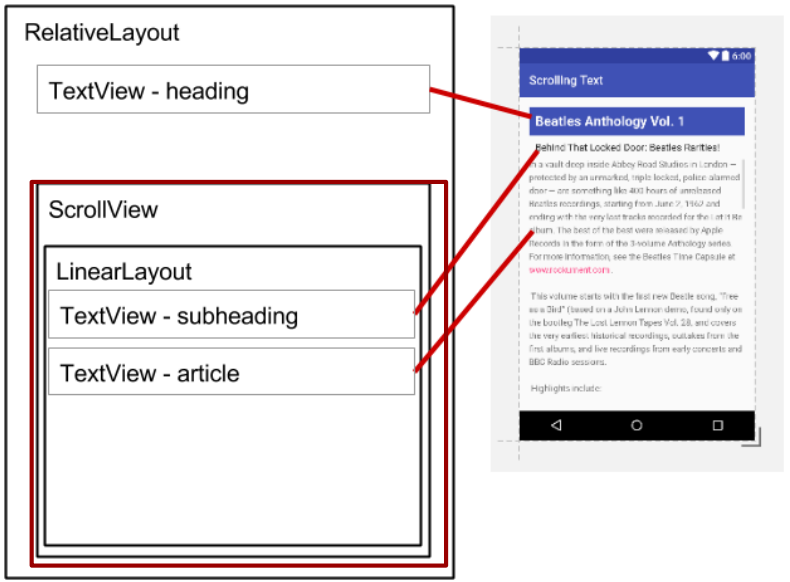
ScrollView: Ejemplo con TextView

```
<ScrollView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@id/article_subheading">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    .../>

</ScrollView>
```



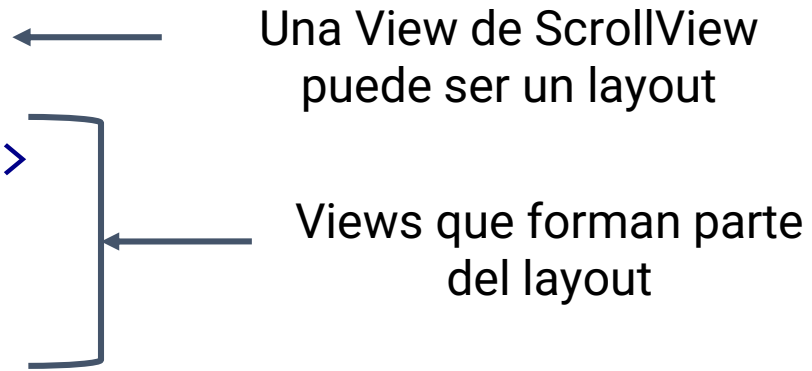
ScrollView: Ejemplo con ViewGroup

```
<ScrollView ...
   <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/article_subheading"
            .../>

        <TextView
            android:id="@+id/article" ... />
   </LinearLayout>
</ScrollView>
```



ScrollView con imagen y botón

```
<ScrollView...>
    <LinearLayout...>
        <ImageView.../>
        <Button.../>
        <TextView.../>
    </LinearLayout>
</ScrollView>
```

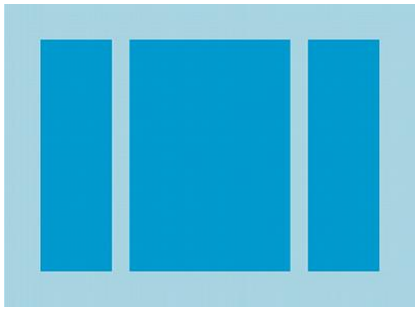


Layout

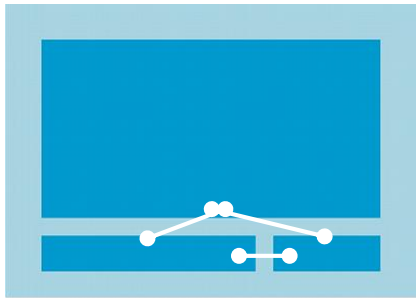
Layout

- Define la estructura de una interfaz de usuario (UI).
- Pueden estar a nivel de fila, columna, rejilla...

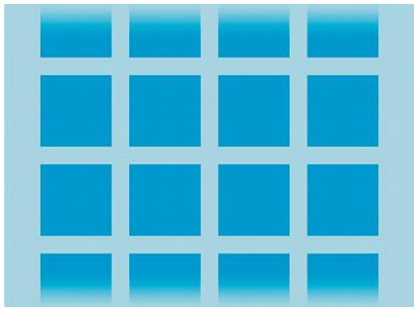
Tipos de Layout más comunes



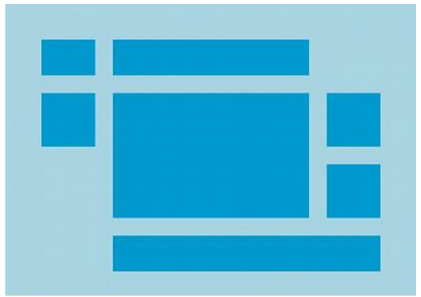
LinearLayout



ConstraintLayout

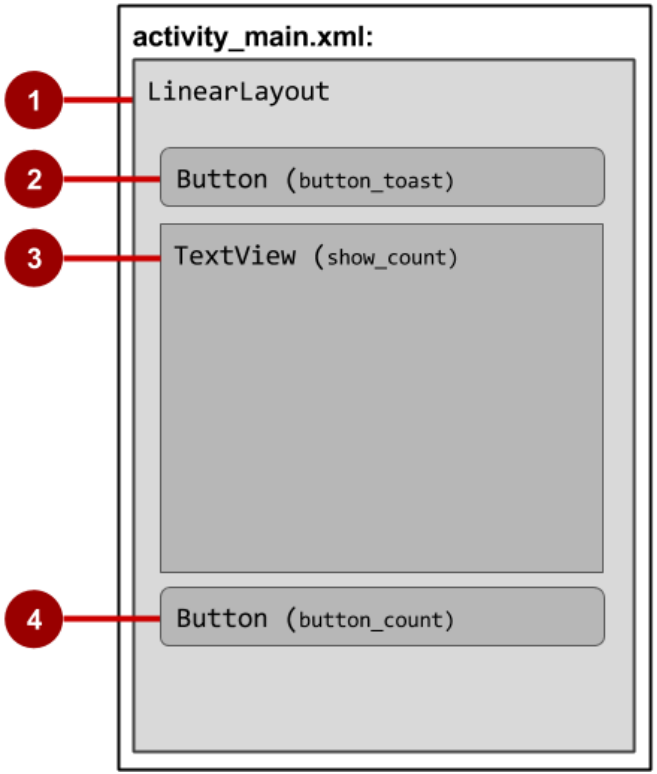


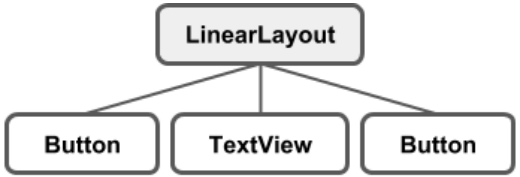
GridLayout

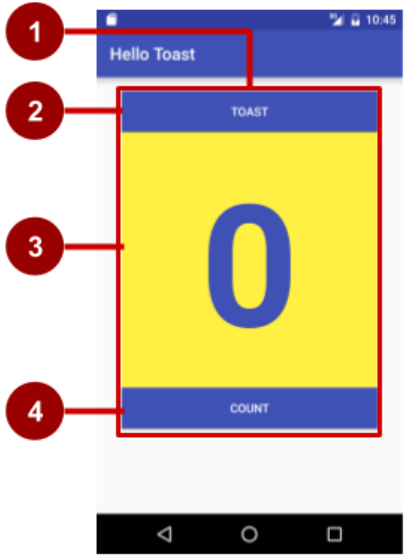


TableLayout

Ejemplo de Layout







Layout: creación a partir XML

```
<LinearLayout
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <Button
       ... />
    <TextView
       ... />
    <Button
       ... />
</LinearLayout>
```

Layout: creación a partir de código Java

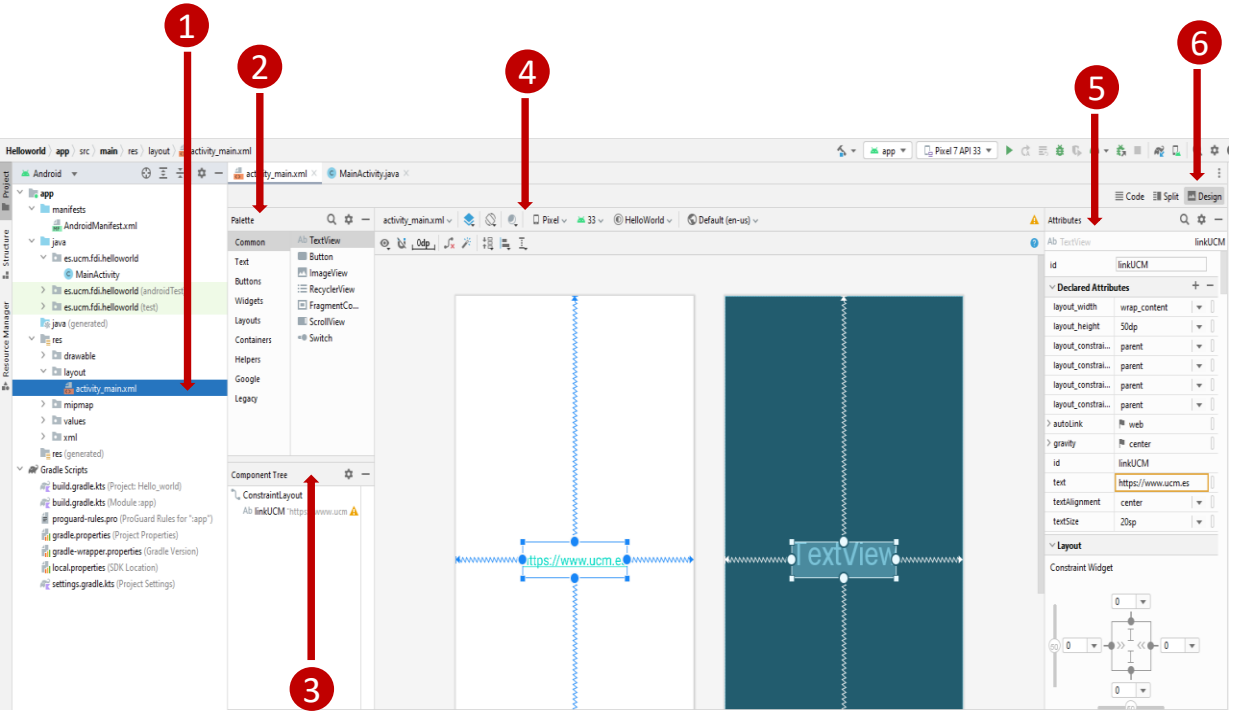
```
LinearLayout linearL = new LinearLayout(this);
linearL.setOrientation(LinearLayout.VERTICAL);

TextView myText = new TextView(this);
myText.setText("¡Hola, Mundo");

linearL.addView(myText);
setContentView(linearL);
```

Establecer anchura y altura con código Java

Establecer **anchura** y **altura** de una *view*:

```
LinearLayout.LayoutParams layoutParams =
  new Linear.LayoutParams(
     LayoutParams.MATCH_PARENT,
     LayoutParams.WRAP_CONTENT);

myView.setLayoutParams(layoutParams);
```

Editor Android Studio para Layout



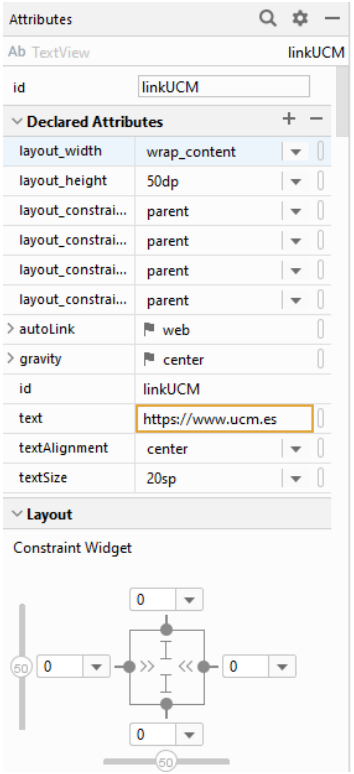
1. Fichero layout XML
2. Panel **Palette**
3. **Component Tree**
4. Barra de herramientas
5. Panel **Attributes**
6. Pestañas **Design**, **Split** y **Code**

Barra de herramientas del editor de layout

1. Selección de los paneles de diseño: Design y Blueprint.
2. Orientación en el editor: Vertical y apaisado.
3. Selector modo noche.
4. Seleccionar el dispositivo para mostrar el diseño.
5. Seleccionar API para mostrar el diseño.
6. Seleccionar tema para el diseño.
7. Seleccionar el idioma por defecto para el diseño.

Panel de atributos

- Hacer clic en la pestaña `Attributes`.
- Seleccionar un elemento a partir de *design*, *blueprint*, o *Component Tree*.
- Modificar los atributos más comunes.

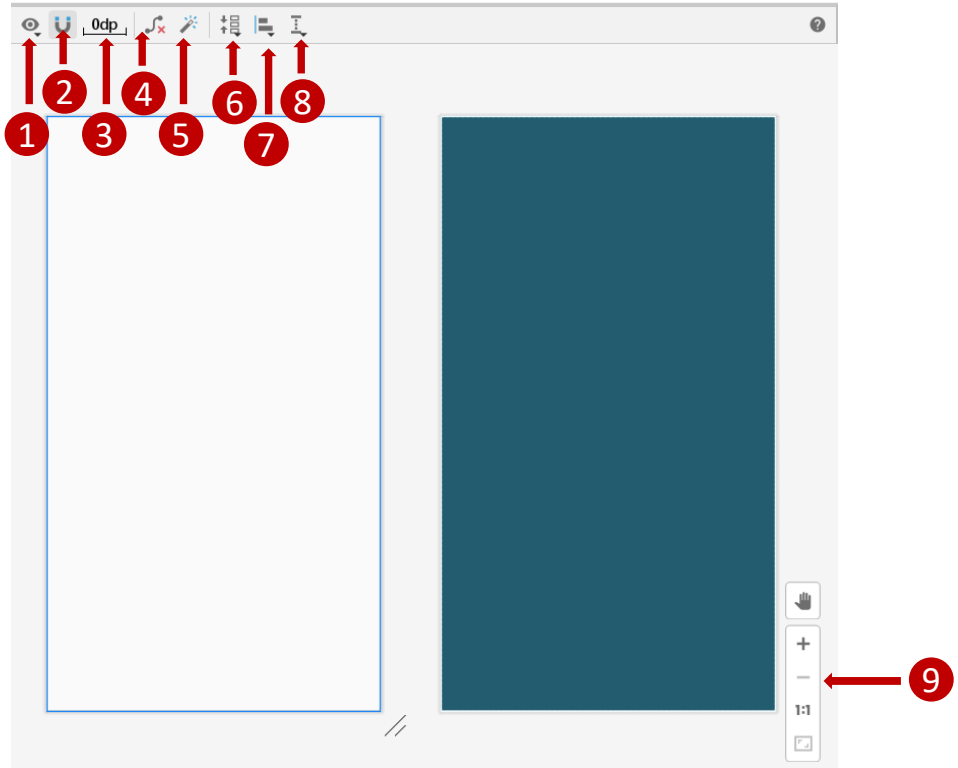


ConstraintLayout

ConstraintLayout

- ConstraintLayout es el layout (diseño) por defecto para un proyecto de Android Studio.
- `ViewGroup` que ofrece flexibilidad para el diseño de layout.
- Proporciona restricciones (en forma de conexiones) para determinar posiciones y alineación de los distintos elementos UI que forman parte del mismo.
- Toda `View` en `ConstraintLayout` va a tener 4 puntos de anclaje: superior, inferior, izquierda y derecha. Desde cada anclaje solamente puede salir una restricción, pero pueden llegar varias.
- Cada `View` ha de tener al menos dos restricciones: una vertical y otra horizontal.
- Solamente se pueden crear restricciones que compartan el mismo plano, ya sea vertical u horizontal.

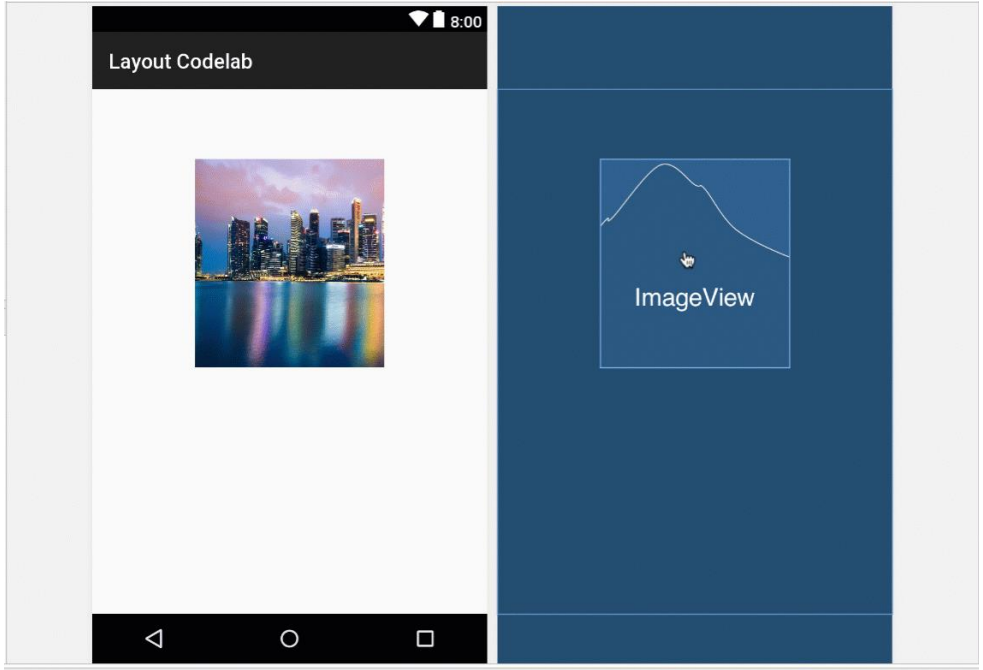
Editor de Layout: Barra de herramientas para ConstraintLayout



1. **Show:** Mostrar restricciones y márgenes.
2. **Autoconnect:** Habilitar o deshabilitar.
3. **Default Margins:** Establecer márgenes por defecto.
4. **Clear All Constraints:** Eliminar todas las restricciones en el layout.
5. **Infer Constraints:** Crear restricciones inferidas.
6. **Pack:** Contraer o expandir los elementos seleccionados.
7. **Align:** Alinear los elementos seleccionados
8. **Guidelines:** Añadir las guías horizontales y verticales.
9. **Zoom**

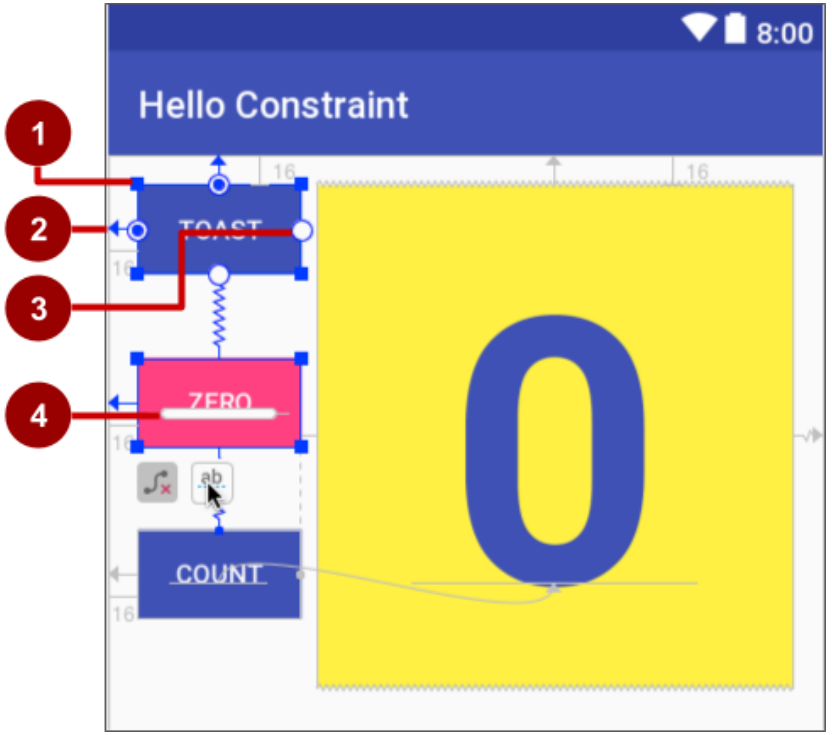
ConstraintLayout: Autoconnect

- Activa Autoconnect en la barra de herramientas.
- Arrastra un elemento a cualquier parte del layout.
- Autoconnect genera automáticamente las restricciones con respecto al layout que lo contiene.



ConstraintLayout: Manejadores

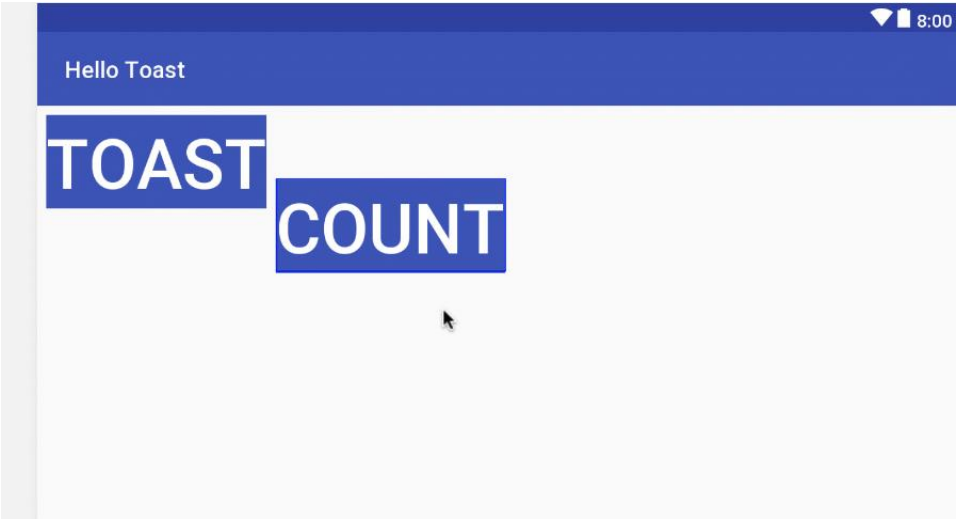
1. Cambiar el tamaño.
2. Línea de restricción y manejador.
3. Manejador de la restricción.
4. Manejador Baseline (botón derecho del ratón => *Show Baseline*).



ConstraintLayout: Alineación con baseline

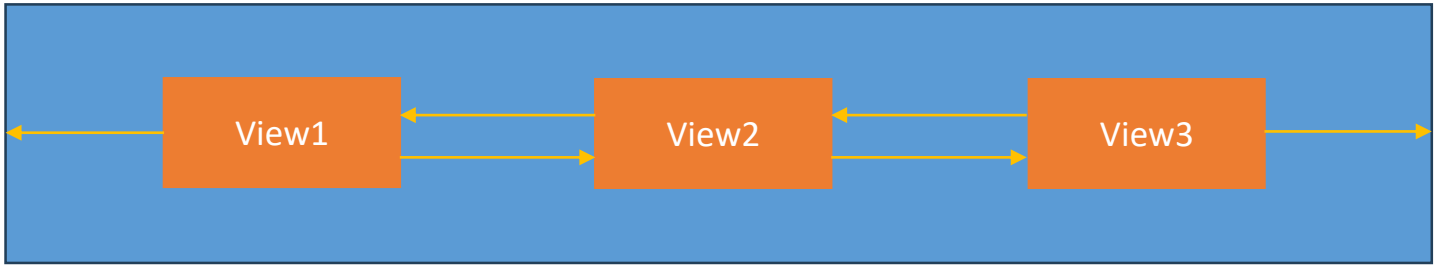
1. Hacer clic en el botón de restricción de baseline 

2. Arrastrar desde esa baseline a la baseline de otro elemento.



ConstraintLayout: Uso de chains

1. Una `chain` es un conjunto de `Views` vinculadas entre sí con restricciones bidireccionales.



2. Se pueden crear cadenas horizontales y verticales.
3. Una `View` puede estar a la vez en una `chain` horizontal y vertical.
4. Utilizar el atributo `chainStyle` asociándolo al primer elemento de la chain. Ejemplo:

```
app:layout_constraintHorizontal_chainStyle="spread"
```

Gestión de eventos

Eventos

- En UI: Hacer clic, pulsar, arrastrar.
- A nivel de dispositivo: DetectedActivity como sería andar, conducir e inclinarse.
- El sistema Android detecta estos eventos.
- Existen manejadores para responder y actuar ante los eventos.

Declarar en XML e implementar en Java

Declarar el manejador asociado a una `View` en el layout XML:

```
android:onClick="showToast"
```

Implementar el manejador con código Java en la `Activity`:

```
import android.view.View;
import android.widget.Toast;
...
public void showToast(View view) {
   String msg = "¡Hola, mundo!";
   Toast toast = Toast.makeText(
     this, msg, Toast.LENGTH_LONG);
   toast.show();
 }
```

Establecer el manejador en Java

```
final Button myButton = (Button) findViewById(R.id.myButton_1);
myButton.setOnClickListener(view -> {
    String msg = "¡Hola, mundo!";
    Toast toast = Toast.makeText(this, msg, Toast.LENGTH_LONG);
    toast.show();
});
```

Recursos

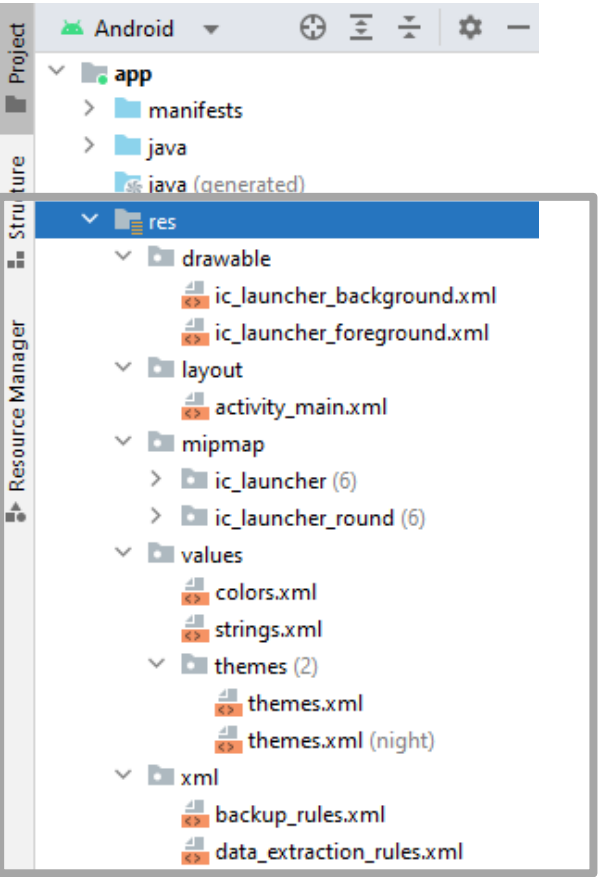
Recursos

Seiri (ordenación).
Seiton (sistematización).
Seiso (limpieza).
Seiketsu (estandarización).
Shutsuke (disciplina).

5S

- Los recursos representan una característica muy importante para el diseño de las apps.
- Permite separar los datos estáticos del código en los layouts.
- Dos categorías: ficheros y valor.
- Strings, dimensiones, imágenes, texto de los menús, colores, estilos.
- Muy útil para la localizar la información.

Ubicación de los recursos en el proyecto



Recursos y ficheros de recursos almcenados en la carpeta **res**

Referencias a los recursos en código

- Layout:

  ```
  setContentView(R.layout.activity_main);
  ```

  O bien:

  ```
  ActivityMainBinding binding = ActivityMainBinding.inflate(getLayoutInflater());
  setContentView(binding.getRoot());
  ```

- View:

  ```
  ReciclerView rv = (RecyclerView) findViewById(R.id.recyclerview);
  ```

  O bien:

  ```
  ReciclerView rv = binding.recyclerview;
  ```

- String:

  Java: `R.string.app_name`

  XML: `android:text="@string/app_name"`

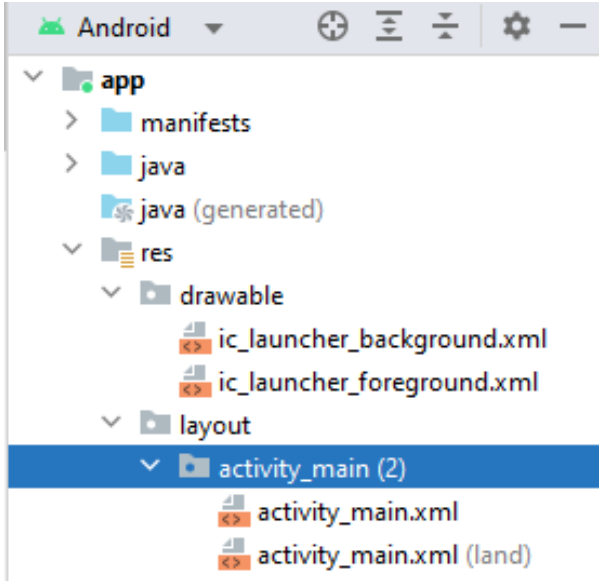
Recursos Alternativos

Variantes de layout

- Los recursos alternativos permiten añadir nuevas configuraciones: orientación del dispositivo, resoluciones de pantalla, distintos idiomas...
- Se identifican a través de lo que se conoce como “qualifier”.
- Android cargará automáticamente los recursos que correspondan (si los encuentra).

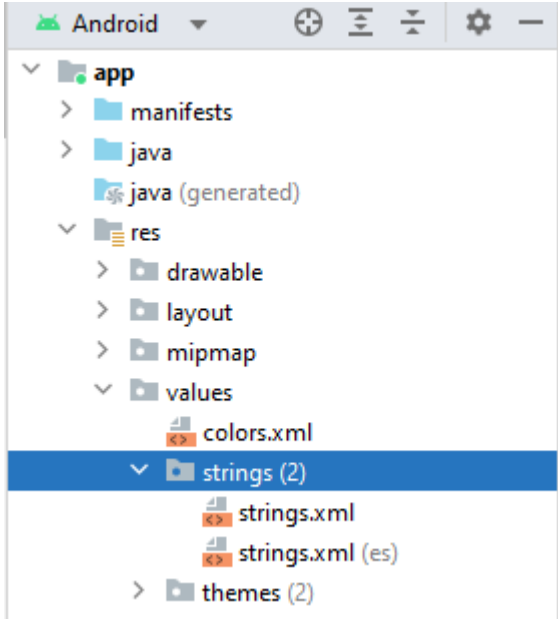
Crear variante para layout apaisado

1. Crear un nuevo fichero de recursos de layout a partir de la carpeta **layout (New > Layout Resource File)**.
2. Poner el mismo nombre de fichero que la variante vertical (ej., activity_main).
3. Seleccionar la variante (qualifier) **Orientation > Landscape**.
4. Se ha creado el fichero para diseñar la variante según convenga: `activity_main.xml (land)`



Crear variante del idioma de los textos

1. Crear un nuevo fichero de recursos de valores a partir de la carpeta **values (New > Values Resource File)**.
2. Poner como nombre **strings.xml**.
3. Seleccionar la variante **Locale > Language: es-Spanish** y **Specific Region Only: Any Region** (ejemplo si se considera el inglés como idioma por defecto de la app y se quiere añadir el idioma español).
4. Se ha creado el fichero para diseñar la variante según convenga: `strings.xml (es)`



Context

¿Qué es context?

- Context es una interfaz de información global relativa al entorno de una aplicación.
- Obtener el *context*:

```
Context context = getApplicationContext();
```

- Una `Activity` es su propio *context*:

```
TextView myText = new TextView(this);
```

O bien:

```
Toast.makeText(MainActivity.this, reply,
               Toast.LENGTH_SHORT).show();
```