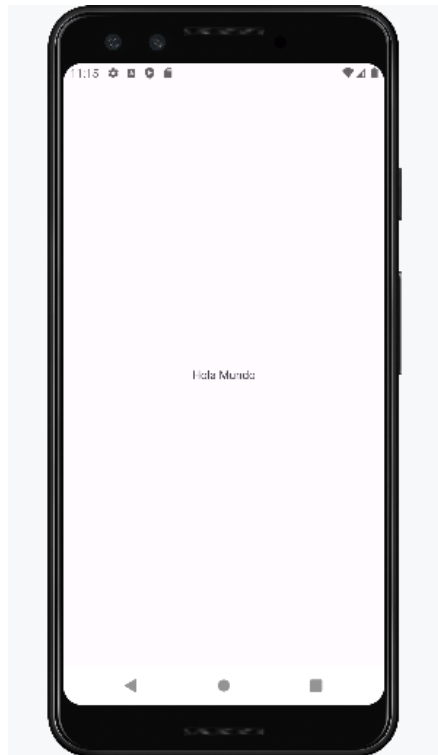


# PROYECTO 1

## HOLA MUNDO



Este proyecto consiste en el desarrollo de nuestra primera app para Android, que simplemente muestra el mensaje “hola mundo” en la pantalla, usando el idioma para el que esté configurado nuestro dispositivo Android.

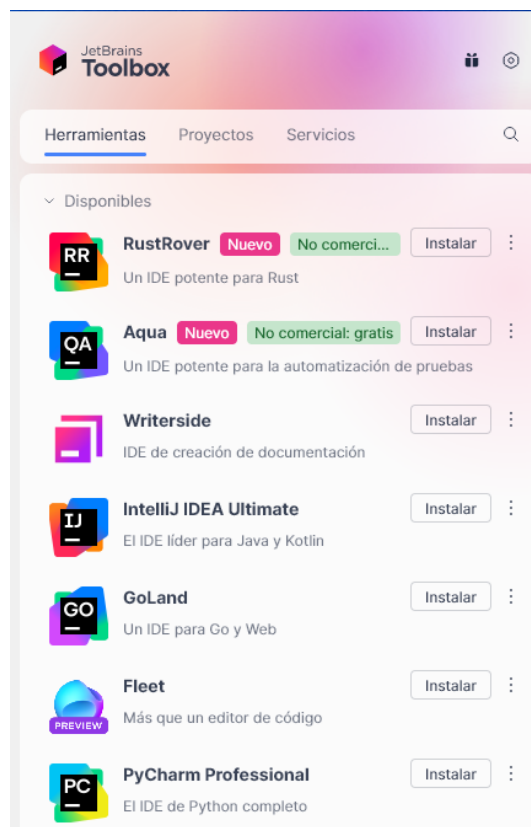
Durante su desarrollo se tratarán estos conceptos:

- Instalación de Android Studio
- Creación de un proyecto
- Estructura de un proyecto
- Concepto de Activity
- Primeros elementos de la interfaz
- Ejecución del proyecto en un emulador
- Ejecución del proyecto en un dispositivo real
- Mensajes de depuración (Logcat)
- Internacionalización
- Soporte para múltiples tipos de dispositivos

## **1 – Instalación de Android Studio**

Android Studio es el IDE recomendado para el desarrollo de aplicaciones nativas para Android. Está basado en el famoso IntelliJ, de la empresa JetBrains. Actualmente, la forma más sencilla de obtenerlo es usando el JetBrains Toolbox

- Busca en Google “**Jetbrains toolbox download**” y entra en el enlace que aparece
- Descarga el **JetBrains ToolBox**
- Cuando termine la descarga, haz doble clic en el instalador y pulsa “siguiente” hasta que esté instalado.
- Una vez que haya sido instalado, verás una ventana como esta



- Pulsa el icono de la lupa y escribe “android studio”. Verás que aparece un resultado en la búsqueda
- Pulsa el botón para instalar Android Studio y acepta los acuerdos de licencia que aparezcan
- Espera hasta que la descarga e instalación finalice.
- Una vez instalado, ejecuta Android Studio pulsando su icono
- La primera vez que se ejecuta, deberán instalarse las herramientas de desarrollo para Android.

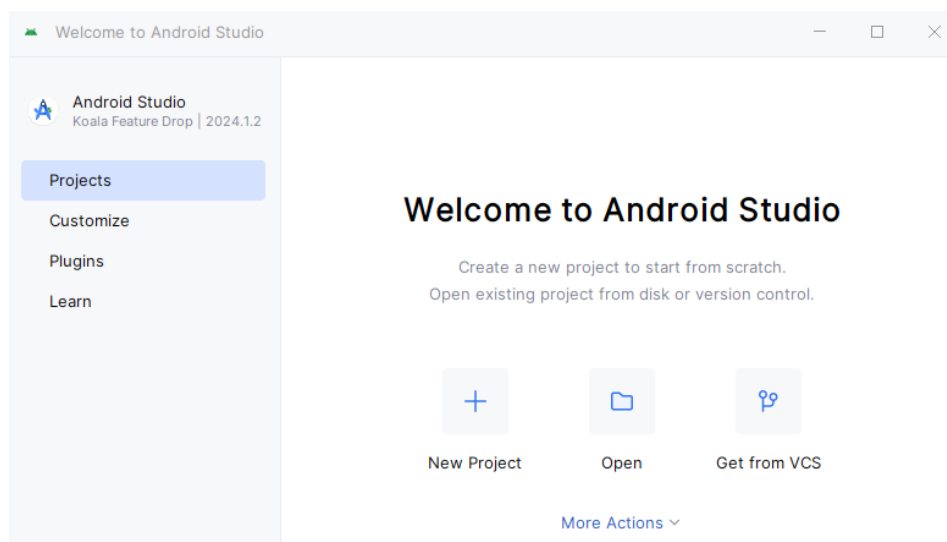
Las herramientas de desarrollo para Android son programas y librerías que utiliza Android Studio internamente para poder desarrollar apps. Entre ellas destacan:

- **Android SDK:** Son las librerías que permiten hacer apps Android. Como se actualizan constantemente, cada SDK va acompañado de un número. El más reciente actualmente es el SDK Level 34
- **Emulador de Android:** Permite ejecutar la app en un emulador en el PC
- Pulsa siguiente y acepta todos los acuerdos de licencia que aparezcan
- Al terminar, si quieres puedes darle a **Customize** y configurar el tema, tamaño de letra, etc.

## 2 – Creación de un proyecto

Cada vez que hacemos una app, crearemos un **proyecto**. El proyecto contiene archivos de código fuente (en Kotlin o Java) y archivos auxiliares (manifiestos, layouts, recursos, etc) que iremos estudiando a lo largo del curso. El proyecto es compilado en un archivo **apk** que puede ser instalado en cualquier dispositivo con el sistema Android

- Abre Android Studio y aparecerá su ventana de inicio



- Pulsa el icono **New Project** para comenzar un nuevo proyecto y aparecerán una serie de plantillas que podemos elegir para comenzar nuestra app.
- Pulsa sobre la plantilla que pone **Empty Views Activity**

Con esta opción, crearemos un proyecto que posee una pantalla con un mensaje que pone “Hello World”

- En el nombre de la app (**Name**), escribe **HolaMundo**
- En el nombre del paquete (**Package Name**) escribe **hlanz.dam.holamundo**
- En **Save location** escribe la ruta donde quieres guardar los archivos del proyecto
- En **Language** selecciona **Kotlin**
- En **Minimum SDK** selecciona el **API 24**

Todos los dispositivos Android tienen instalada una versión de la librería de Android (lo que antes hemos visto que era el SDK). Con el parámetro **Minimum SDK** estamos eligiendo la versión de la librería de Android que vamos a utilizar, lo que en realidad indica que los móviles que no alcancen esa versión, no podrán usar la app.

- Pulsa **Finish** cuando termines y tras unos momentos (pueden ser minutos según el equipo) se abrirá la ventana del proyecto. También puede ocurrir que tengan que descargarse componentes que aún no estén instalados.

☞ Android Studio necesita un equipo bastante potente para funcionar correctamente. Los requisitos mínimos exigen un procesador de 64 bits, mucha ram (de 8 GB en adelante) y bastante espacio en disco duro (los emuladores e imágenes de Android ocupan mucho espacio).

- Si todo ha ido bien, se abrirá la ventana principal Android Studio mostrando la estructura del proyecto y código fuente que ha generado por nosotros. Antes de trabajar con él, es conveniente asegurarse de que en la barra inferior de la pantalla no se ve ninguna barra de progreso haciendo cosas en segundo plano.

### **3 – Estructura del proyecto**

En la zona de la izquierda, podemos ver la **estructura del proyecto**, que son las carpetas que contienen los archivos que formarán nuestra app. Dichas carpetas son:

- **Manifiesto:** La carpeta **Manifests** Contiene un archivo llamado **AndroidManifest.xml**, que usaremos más adelante para indicar al sistema Android cómo es nuestra app, los permisos que necesita, etc.
- **Código fuente:** La carpeta **kotlin+java** contiene los paquetes donde se guardará el código fuente de la app y los tests de las clases que sean necesarios
- **Recursos:** La carpeta **res** guarda archivos auxiliares que son usados por la app. Por ejemplo, imágenes, sonidos, diseños de las pantallas, descripción de los temas, colores usados por la app, etc.
- **Configuración:** Los proyectos de Android Studio siguen el sistema de construcción **Gradle**, así que hay una carpeta donde podremos encontrar el **build.gradle.kts**, que usaremos para descargar las librerías externas que necesitemos.


- Abre el archivo **AndroidManifest.xml** y observa su contenido. No pasa nada que en este momento no entiendas lo que significa cada cosa. Simplemente observa que aparecen definidas cosas como el nombre de la app (**android:label**), el icono que utilizará (**android:icon**), el tema (o estilo) usado (**android:theme**) o si el texto se mostrará de derecha a izquierda en los países con dichos alfabetos (**android:supportsRtl**)
- Abre la carpeta **Kotlin+java** y el paquete **hlanz.dam.holamundo**. Verás un archivo llamado **MainActivity**.

#### **4 – Concepto de Activity**

En Android una **Activity** es una ventana de nuestra app. Por cada Activity, hay dos archivos:

- **Archivo de diseño:** Es un archivo en formato xml que se encuentra en la carpeta **res/layout** donde se diseña, de forma parecida al HTML usando etiquetas, el contenido de la ventana.
- **Archivo de código fuente:** Es un archivo de código Kotlin (.kt) o Java (.java) que se encuentra en la carpeta **kotlin+java**. En este archivo se programan las acciones que se producirán cuando el usuario interactúe con la ventana, y también se configura el comportamiento de la ventana cuando la app es inicializada, minimizada, cerrada, etc.

Cuando creamos el proyecto usando **empty views activity**, nuestro proyecto comienza con un Activity llamado **MainActivity**, que es la pantalla que se abre cuando se pone en marcha la app.

- Abre la carpeta **res/layout** y haz click sobre **activity\_main.xml**.
- Busca en la pantalla estos botones : 
- Pulsa los botones anteriores y alterna entre las vistas de código, diseñador y ambas a la vez.

Verás que:

- En la vista de código se muestra el código fuente de la ventana (diseñado con etiquetas, como si fuera una especie de HTML propio de Android)
- En la vista de diseño, se ve, de forma gráfica, cómo quedaría la ventana en un dispositivo móvil
- En la vista **split** se muestran las dos vistas anteriores a la vez, y es la opción recomendada cuando disponemos de un monitor de alta resolución.

- Haz clic sobre él y se abrirá la pantalla de código fuente del archivo **MainActivity.kt**. Veremos código fuente escrito en lenguaje Kotlin que ha escrito Android Studio por nosotros. Normalmente usaremos ese código como plantilla y no tocaremos lo que ya hay escrito, sino que nos dedicaremos a añadir nuevas cosas.

En el código fuente destacamos lo siguiente:

- Nuestra **ActivityMain** es una clase, que **hereda** (recibe métodos) de una clase padre llamada **AppCompatActivity**
- Hay un método llamado **onCreate**, donde está programado todo lo que nos ha escrito Android Studio. Ese método es llamado cuando se inicia la app, y contiene el código que se ejecutará en dicho momento.
- La llamada a **setContentView** hace que MainActivity utilice el diseño definido en **activity\_main.xml**.

- El código que nos genera Android Studio está optimizado para las últimas versiones de los móviles Android. Si queremos que funcione de forma óptima también en tablets (como luego veremos), vamos a borrar algunas de las cosas que nos ha añadido, de forma que todo se quede así:

```
1. class MainActivity : AppCompatActivity() {
2.     override fun onCreate(savedInstanceState: Bundle?) {
3.         super.onCreate(savedInstanceState)
4.         setContentView(R.layout.activity_main)
5.     }
6. }
```

## **5 – Primeros elementos de la interfaz**

- Abre el archivo **activity\_main.xml** y comprueba que hay un elemento padre llamado **ConstraintLayout**, que contiene un elemento hijo llamado **TextView**

- **ConstraintLayout:** En Android hay varios tipos de **layout**. Un layout es un elemento padre que estudiaremos más adelante. Su misión es contener elementos de la interfaz (que se llaman **Vistas**) y darles una distribución, como por ejemplo poner cada uno a continuación de otro, ponerlos en forma de tabla, etc. Un ConstraintLayout es el tipo de layout que se usa cuando se diseña gráficamente la interfaz usando el diseñador de Android Studio.
- **TextView:** Es una etiqueta de texto

- Observa que las etiquetas que abren esos elementos poseen atributos

- **android:id:** Es un atributo que sirve para poner un identificador único al elemento de la interfaz. Siempre comienza por **@+id/** y luego tiene el nombre del identificador.
- **android:layout\_width:** Es el ancho del elemento. Puede ser:
  - **match\_parent:** El elemento tiene el mismo ancho que su contenedor
  - **wrap\_content:** El elemento tiene la anchura mínima para mostrar su contenido
- **android:layout\_height:** Es la altura del elemento, y admite los dos valores que hemos visto para la anchura.


- Cambia el valor del atributo **android:text** y ponle “Hola Mundo”


El atributo **android:text** se encuentra en muchos elementos (cuadros de texto, botones, etc) y contiene el texto que se mostrará dentro del elemento.

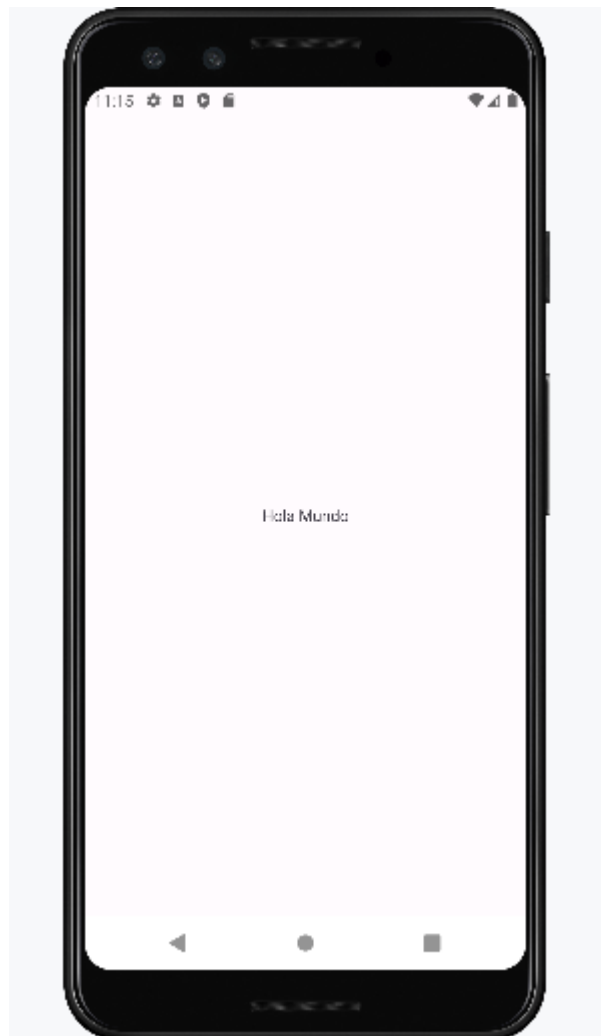
## **6 – Ejecución del proyecto en un emulador**

Los proyectos de Android Studio se pueden poner en marcha en un emulador o en un dispositivo real conectado al PC por un cable USB.

Para compilar y poner en marcha el proyecto, pulsaremos el botón de **play**. Pero para que funcione, primero deberemos descargar y configurar un emulador del dispositivo Android que queramos emular.

- En la derecha, pulsa el icono **Device Manager** 
- Pulsa el botón **+** para añadir un nuevo dispositivo
- Pulsa **create virtual device**
- En la categoría **Phone** podemos añadir el emulador de un smartphone. Vamos a añadir, por ejemplo, el **Pixel 8 Pro** y después pulsa **Next**
- En la siguiente pantalla elegimos la versión de las librerías de Android que va a llevar ese móvil. Por ejemplo, vamos a añadir la versión **32** y pulsamos **Next**. Es posible que deba conectarse a Internet para descargar la imagen.
- Cuando todo esté terminado, en la última pantalla revisamos que todo es correcto y pulsamos **Finish**.

- Una vez que ya tenemos el emulador, podemos ejecutar el proyecto pulsando el icono de play 



### **7 – Ejecución del proyecto en un dispositivo real**

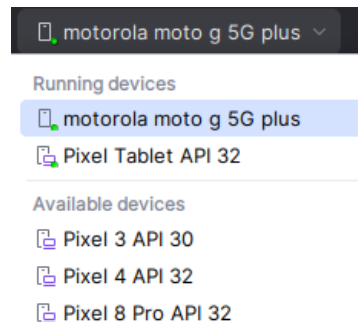
Android Studio permite ejecutar el proyecto en un dispositivo real conectado por cable USB al PC donde estemos desarrollando la app. Dicho dispositivo debe tener activado el **modo para desarrolladores**, o si no, no tendremos permiso para probar la app


- Coge un móvil con sistema Android
- Vete a la pantalla de **preferencias**
- Pincha en **acerca del teléfono**
- Busca el **número de compilación** y pulsa 7 veces sobre él. Si lo haces bien, aparecerá un mensaje diciendo que las opciones para desarrolladores están activadas.





- A continuación, conecta el móvil al PC con un cable USB y en la ventana que aparece, permite la **depuración USB**
- Abre el proyecto que estamos haciendo en Android Studio
- En la barra de arriba, busca la lista desplegable en la que podemos elegir el dispositivo (virtual o real) en el que vamos a ejecutar la app.




- Elige tu móvil y ejecuta la app con el botón . Si lo haces bien, la app se descargará e instalará en tu móvil y se ejecutará.

## **8 – Mensajes de depuración (Logcat)**

Cuando programamos una app, es muy frecuente que en el código fuente queramos mostrar mensajes internos para saber el contenido de variables, ver si el programa entra en una zona concreta, etc.

Android posee un sistema común llamado **Logcat** donde todas las apps pueden volcar mensajes. Cada mensaje tiene un texto y una etiqueta que nos permite reconocerlo de entre todos los mensajes que las apps vuelcan en Logcat.

Para mostrar un mensaje llamaremos al método **Log.d** y pasaremos en primer lugar, la etiqueta que permite buscar el mensaje en Logcat, y luego, el texto del mensaje.

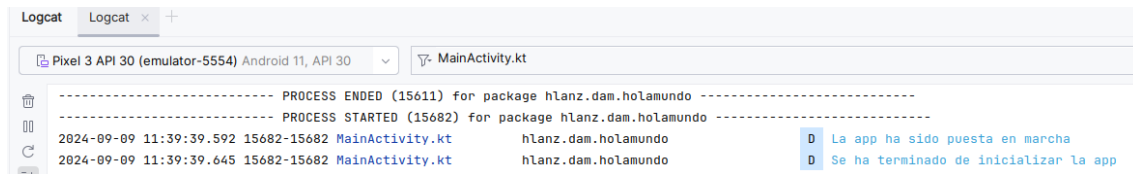
- Para abrir la ventana Logcat, pulsa el icono 
- Ejecuta la app y verás como la ventana de Logcat se llena con un montón de mensajes. Esos mensajes los están escribiendo las librerías de Android que utiliza internamente nuestra app y no les haremos caso.
- Abre el archivo **MainActivity** y justo al inicio y al final del método **onCreate** vamos a volcar dos mensajes, así:

```

1. class MainActivity : AppCompatActivity() {
2.     override fun onCreate(savedInstanceState: Bundle?) {
3.         Log.d("MainActivity.kt", "La app ha sido puesta en marcha")
4.         super.onCreate(savedInstanceState)
5.         setContentView(R.layout.activity_main)
6.         Log.d("MainActivity.kt", "Se ha terminado de inicializar la app")
7.     }
8. }

```

- Ejecuta la app y vamos a comprobar que los mensajes que hemos puesto aparecen en Logcat. Para ello, en el buscador escribe **MainActivity** (que es la etiqueta que hemos puesto a los mensajes) y comprueba que aparecen ambos mensajes



- Para no tener que repetir constantemente la etiqueta **"MainActivity.kt"**, es costumbre definir al inicio del archivo una **constante** con dicho valor, y referenciarla al llamar al método **Log.d**, así:

```

1. const val TAG = "MainActivity.kt"
2. class MainActivity : AppCompatActivity() {
3.     override fun onCreate(savedInstanceState: Bundle?) {
4.         Log.d(TAG, "La app ha sido puesta en marcha")
5.         super.onCreate(savedInstanceState)
6.         setContentView(R.layout.activity_main)
7.         Log.d(TAG, "Se ha terminado de inicializar la app")
8.     }
9. }

```

## 9 – Internacionalización

El mensaje **Hola Mundo** de nuestra app está escrito en lenguaje español, pero las apps profesionales se suben a la **Play Store** y todo el mundo puede descargarlas. Android dispone de un sistema que permite cambiar los mensajes según el idioma con el que esté configurado el teléfono. A fecha de hoy, la traducción no es automática, y debemos, por cada idioma soportado, crear un archivo en la carpeta **res/values** que contenga los mensajes. Cada mensaje tendrá un **identificador**, que colocaremos en el código fuente o en el archivo de diseño de la interfaz.

- Abre el archivo **activity\_main.xml** y observa que el atributo **android:text** del **TextView** contiene el mensaje **Hola Mundo** escrito “a mano” en español
- Abre la carpeta **res/values** y abre el archivo llamado **strings.xml**.

Este archivo contiene los mensajes que la app usará por defecto

- Vamos a agregar un mensaje con el identificador **saludo** cuyo texto será **Hola Mundo**. Para ello, simplemente añade la siguiente línea a **strings.xml**

```

1. <resources>
2.     <string name="app_name">HolaMundo</string>
3.     <string name="saludo">Hola Mundo</string>
4. </resources>

```

- Ahora vuelve al archivo **activity\_main.xml** y cambia el valor del atributo **android:text** del **TextView** para que haga referencia al mensaje cuyo identificador es **saludo**, de esta forma:

```

1. <TextView
2.     android:layout_width="wrap_content"
3.     android:layout_height="wrap_content"
4.     android:text="@string/saludo"
5.     app:layout_constraintBottom_toBottomOf="parent"
6.     app:layout_constraintEnd_toEndOf="parent"
7.     app:layout_constraintStart_toStartOf="parent"
8.     app:layout_constraintTop_toTopOf="parent" />

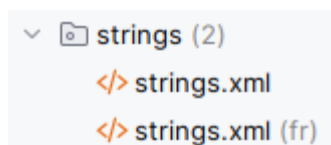
```

Cuando el atributo **android:text** comienza por **@string/** significa que el texto del elemento no es un mensaje escrito “a mano”, sino que se toma del archivo **strings.xml**

- Ejecuta la app y comprueba que todo se sigue viendo igual que antes, aunque ahora, internamente, el mensaje **Hola Mundo** está siendo obtenido del archivo **strings.xml**
- Vamos ahora a añadir un archivo **strings.xml** para el idioma francés. Para ello, pulsa el botón derecho del ratón sobre la carpeta **/res/values** y elige **new → values resource file**
- En **File name** escribe **strings.xml**
- En **Available qualifiers** elige **Locale** y pulsa el botón >>
- En **Language** elige **fr:French** (*puedes pulsar la tecla f, luego la r y verás que se abre un buscador que te permite elegirlo fácilmente*)

Observa que puedes concretar más eligiendo un país concreto de habla francesa, como Francia, Bélgica, Gabón, etc. Eso hará que el móvil muestre mensajes apropiados a la variación del francés del idioma elegido.

- Pulsa **OK** y observa que se crea un segundo archivo **strings.xml**, que aparece con una indicación **(fr)** a su lado, que indica que será usado cuando el móvil esté configurado para el idioma francés.



- En el archivo que se abre, añade el mensaje con identificador **saludo** y su texto es **Bonjour le monde**, de esta forma:

```

1. <!--?xml version="1.0" encoding="utf-8"?-->
2. <resources>
3.     <string name="saludo">Bonjour le monde</string>
4. </resources>

```

- Para comprobar que funciona, inicia el emulador y cambia el idioma del móvil a francés. *Esto se hace de forma diferente según el móvil, pero lo habitual es ir a los ajustes (**settings**) y buscar **Languages & input**. Allí podremos añadir un nuevo lenguaje y alternar entre todos los que tengamos instalados.*
- Comprueba que al cambiar el lenguaje del móvil, la Activity se reinicia, mostrando el mensaje actualizado. *Puedes comprobar el reinicio viendo que en Logcat se muestran los dos mensajes que pusimos en el apartado anterior.*

La Activity se reinicia porque Android detecta un **cambio estructural**. Un cambio estructural es un cambio “gordo” que afecta a la aplicación (rotar el móvil, cambiar el idioma, conectar un teclado, etc). Cada vez que hay un cambio estructural, la Activity es reiniciada, con lo que se vuelve a crear desde el principio, ya con el nuevo cambio en la configuración. Por ese motivo, el archivo **activity\_main.xml** es recargado y el mensaje se muestra correctamente en el nuevo idioma.

Cuando se programa en Android, es muy importante detectar y saber responder a los cambios estructurales, ya que los datos que estén en la memoria de la Activity se pierden

- Actualiza el mensaje de Logcat haciendo que se muestre el idioma con el que está configurado el móvil, de esta forma:

```
const val TAG = "MainActivity.kt"
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        Log.d(TAG, "La app ha sido puesta en marcha con idioma ${Locale.getDefault()}")
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.d(TAG, "Se ha terminado de inicializar la app")
    }
}
```

- Comprueba que al cambiar el idioma del móvil, el mensaje de Logcat muestra el idioma

## **10 – Soporte para múltiples tipos de dispositivos**

La aplicación que hemos hecho tiene la misma apariencia en un móvil que en un tablet. De manera parecida al idioma, podemos añadir un segundo **activity\_main.xml** para los tablet, de forma que en un tablet la interfaz se vea diferente.

Aunque añadamos un **activity\_main.xml** propio para tablets, el comportamiento de la app será el mismo que en el móvil, debido a que el archivo **MainActivity** seguirá siendo el mismo en ambos.

- Pulsa el botón derecho del ratón en **res/layout** y elige **new → layout resource file**
- En **File name** escribe **activity\_main**

- En **Available qualifiers** elige **Size** y pulsa el botón >>
- En **Screen size** elige **Large** y pulsa **OK**.

Con esto, estamos creando un nuevo **activity\_main.xml** que se utilizará cuando Android detecte una pantalla de tamaño grande, como una tablet

- Haz doble clic sobre el nuevo archivo **activity\_main.xml** que ha aparecido (*verás que pone **large** para indicar que es la versión del archivo para pantallas grandes*)
- Activa la vista de código fuente y borra todo lo que haya en él
- Vete al **activity\_main.xml** para móvil, copia todo su contenido y pégalo en el **activity\_main.xml** que estás haciendo para la tablet.
- Añade al **TextView** el atributo **android:textColor**, de forma que el color de texto sea un color llamado **rojo**, así:

```

1. <TextView
2.     android:layout_width="wrap_content"
3.     android:layout_height="wrap_content"
4.     android:text="@string/saludo"
5.     android:textColor="@color/rojo"
6.     app:layout_constraintBottom_toBottomOf="parent"
7.     app:layout_constraintEnd_toEndOf="parent"
8.     app:layout_constraintStart_toStartOf="parent"
9.     app:layout_constraintTop_toTopOf="parent" />

```

De la misma forma que sucede con los mensajes, los colores también se definen mediante nombres, en el archivo **/res/values/colors.xml**

- Ahora tenemos que ir al archivo **res/values/colors.xml** para definir allí un color llamado **rojo**, de esta forma:

```

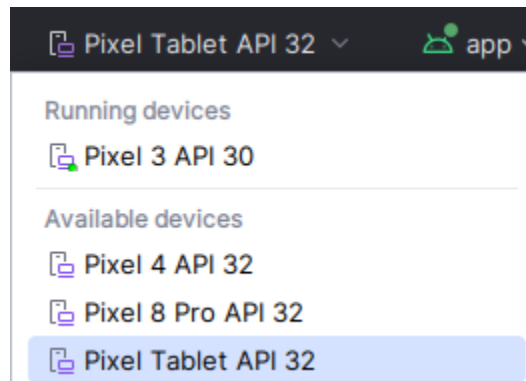
1. <!--?xml version="1.0" encoding="utf-8"?-->
2. <resources>
3.     <color name="black">#FF000000</color>
4.     <color name="white">#FFFFFFF</color>
5.     <color name="rojo">#FFF0000</color>
6. </resources>

```

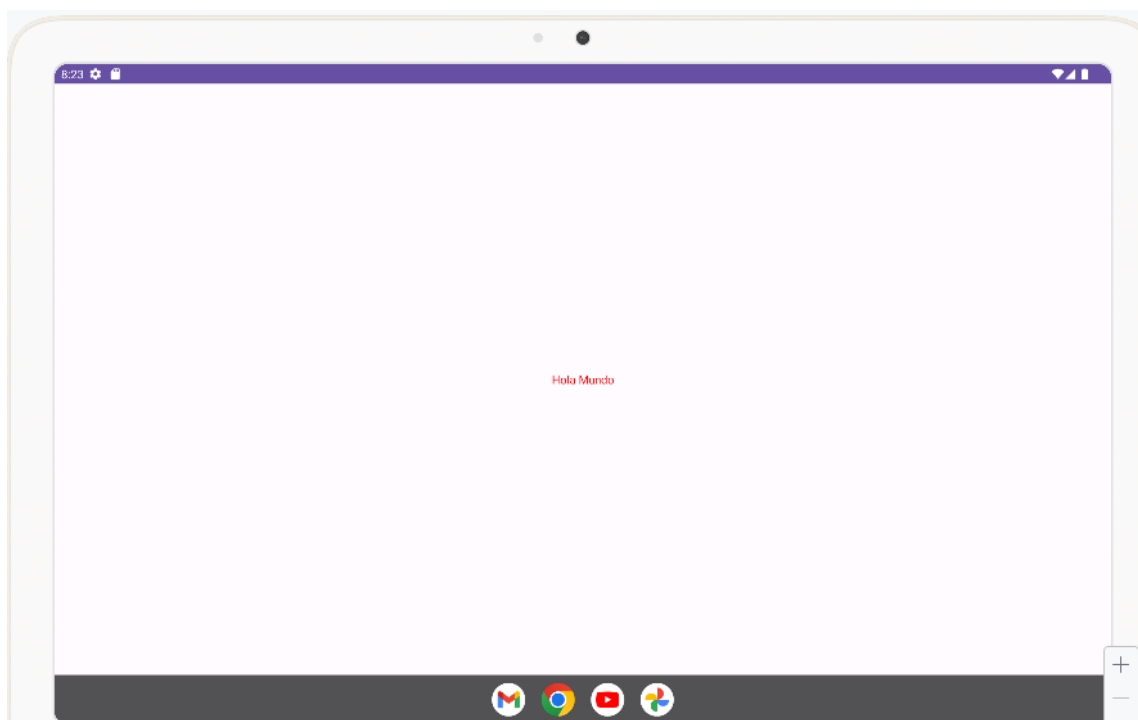
Los colores se definen igual que en el HTML, en formato #RGB usando dos valores hexadecimales para indicar la cantidad de cada color. También se admite el formato #ARGB, siendo A el nivel de transparencia

- Abre el **Device Manager** y añade un emulador para la tablet **Pixel Tablet** que use la versión 32 de las librerías de Android

- En la zona superior de la ventana, justo a la derecha del botón de play, verás una lista desplegable que permite elegir el dispositivo donde deseas ejecutar la app. Elige la **Pixel Tablet API 32** que acabas de crear



- Ejecuta la aplicación en la **Pixel Tablet** y comprueba que se muestra con el **activity\_main.xml** que muestra en rojo el mensaje **Hola Mundo**. *Es posible que tengas que ir a la pestaña **Running Devices** y finalizar el proceso con el móvil*



## **11 – Ejercicios**

Ejercicio 1: Sobre la app HolaMundo, abre el archivo **/res/values/themes/themes.xml** comprueba que el estilo pulsado por la app y comprueba si el atributo **parent** tiene alguno de estos valores:

- **Theme.Material3.DayNight.NoActionBar**
- **Theme.MaterialComponents.DayNight.DarkActionBar**

Cambia el que tenga puesto por el otro. Ejecuta la app y comprueba lo que sucede.

Ejercicio 2: Sobre la app HolaMundo, tras haber realizado el ejercicio anterior, abre el archivo **AndroidManifest.xml** y busca en él el lugar donde aparece el nombre de la app. Cámbialo y pone a la app el nombre **Hola Mundo App**. Lanza la app y comprueba que el nombre se muestra correctamente en la barra de la app.

Ejercicio 3: Traduce la app HolaMundo a los siguientes idiomas: inglés, alemán, italiano, portugués y holandés (busca en Google la traducción de los mensajes a todos los idiomas). No olvides traducir también el nombre de la app que has puesto en el ejercicio 2.

Ejercicio 4: Haz que en el idioma inglés de EEUU el mensaje que muestre la app sea “Hi everybody” en lugar de “Hello world”, que debería ser el mensaje que se muestre en el resto de países de habla inglesa. Comprueba que la app funciona correctamente.

Ejercicio 5: Haz que la app muestre la barra del ejercicio 2 solo para móviles, mientras que para tablets no se muestre dicha barra.

*Pista: Haz un archivo **themes.xml** específico para tablets*

Ejercicio 6: Añade al archivo **/res/values/colors.xml** un color llamado **rojo** con el color amarillo (busca en Internet su RGB si no lo sabes). A continuación, vete al archivo **values/themes.xml** y dentro de la etiqueta **<style>** añade esta línea:

```
<item name="colorPrimary">@color/rojo</item>
```

Ejecuta la app y comprueba si se produce algún cambio apreciable.