

UD3.1 – Introducción a Android

2º CFGS
Desarrollo de Aplicaciones Multiplataforma
2023-24

1.- Introducción

La experiencia de uso en las aplicaciones móviles y en las aplicaciones de escritorio es muy diferente.

En una aplicación móvil la interacción del usuario **no empieza siempre en el mismo lugar**.

Por ejemplo:

- Si se abre la aplicación de correo electrónico lo más habitual es que se muestre la bandeja de entrada o la última ventana abierta en la aplicación.
- Si se está navegando por una página web y se pulsa el botón de contacto para enviar un mail, es probable que se abra la aplicación de correo pero directamente para escribir el correo electrónico.

1.- Introducción

En las aplicaciones de escritorio el punto de inicio de la aplicación es el método **main** que incluye el código que se ejecuta al iniciar la aplicación.

En las aplicaciones móviles no se puede realizar de la misma manera debido a que se puede iniciar una aplicación en diferentes puntos de la misma como se ha visto en el punto anterior.

Es por ello que el **ciclo de vida** de las aplicaciones móviles es distinto al de las aplicaciones de escritorio.

2.- Activity

La clase **Activity** es un componente crucial en una aplicación Android.

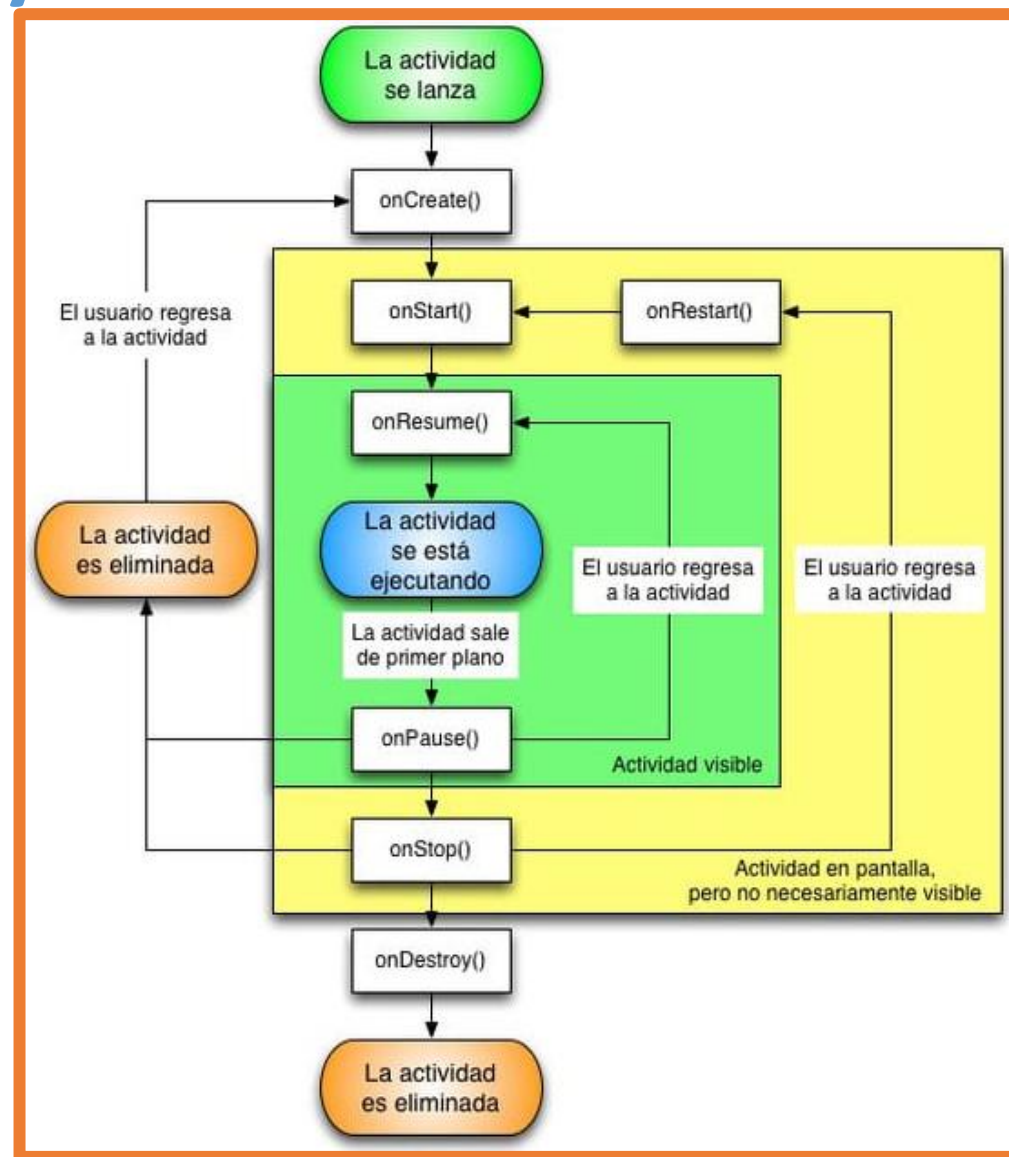
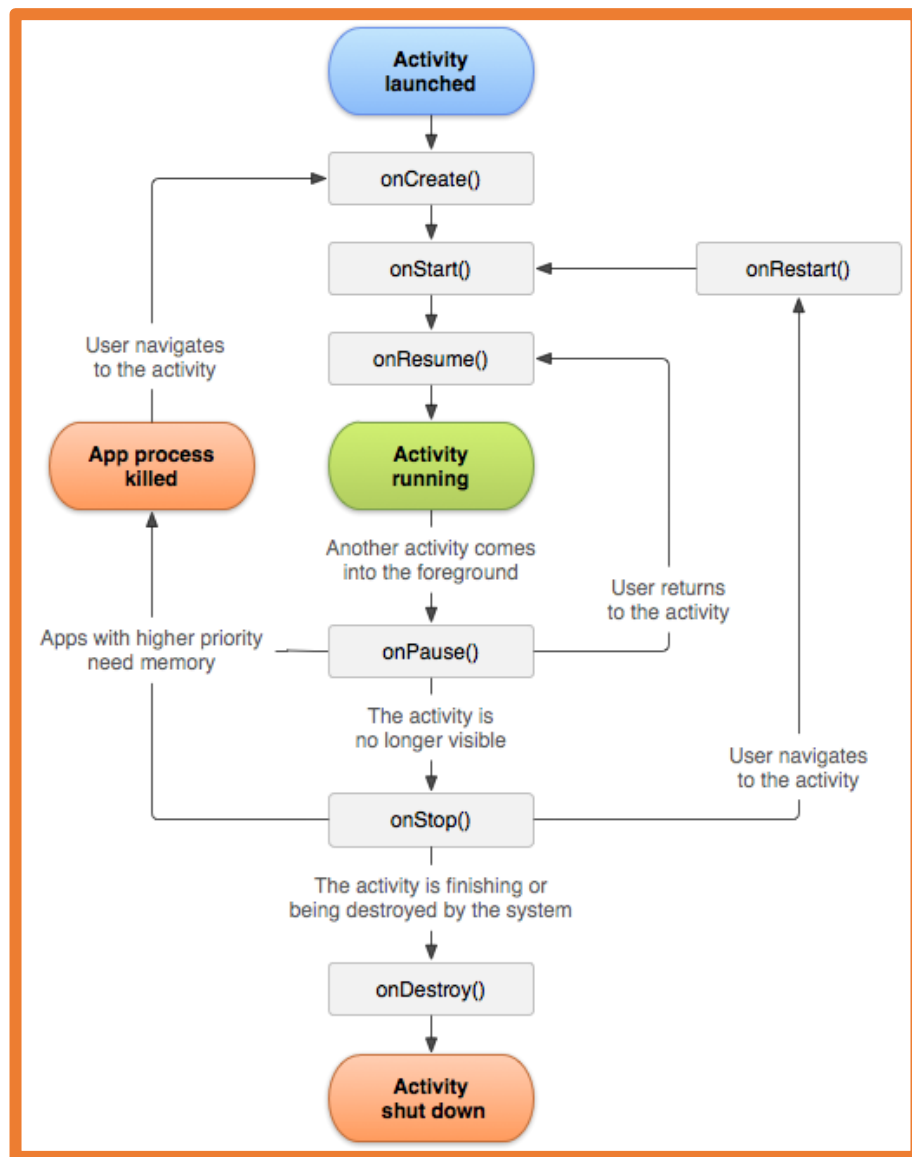
En Android **cada pantalla de la aplicación está definida en una Activity.**

Una **Activity** es el **punto de entrada** para la interacción del usuario con la aplicación.

En Android el código que inicia una **Activity** corresponde a una llamada a un **método** que corresponde a una **de las etapas específicas del ciclo de vida** de la **Activity**.

Conforme el usuario navega, sale y regresa a la aplicación, las diferentes **Activities** de la aplicación pasan por diferentes **estados** de su **ciclo de vida** (lifecycle).

3.- Ciclo de vida de una Activity



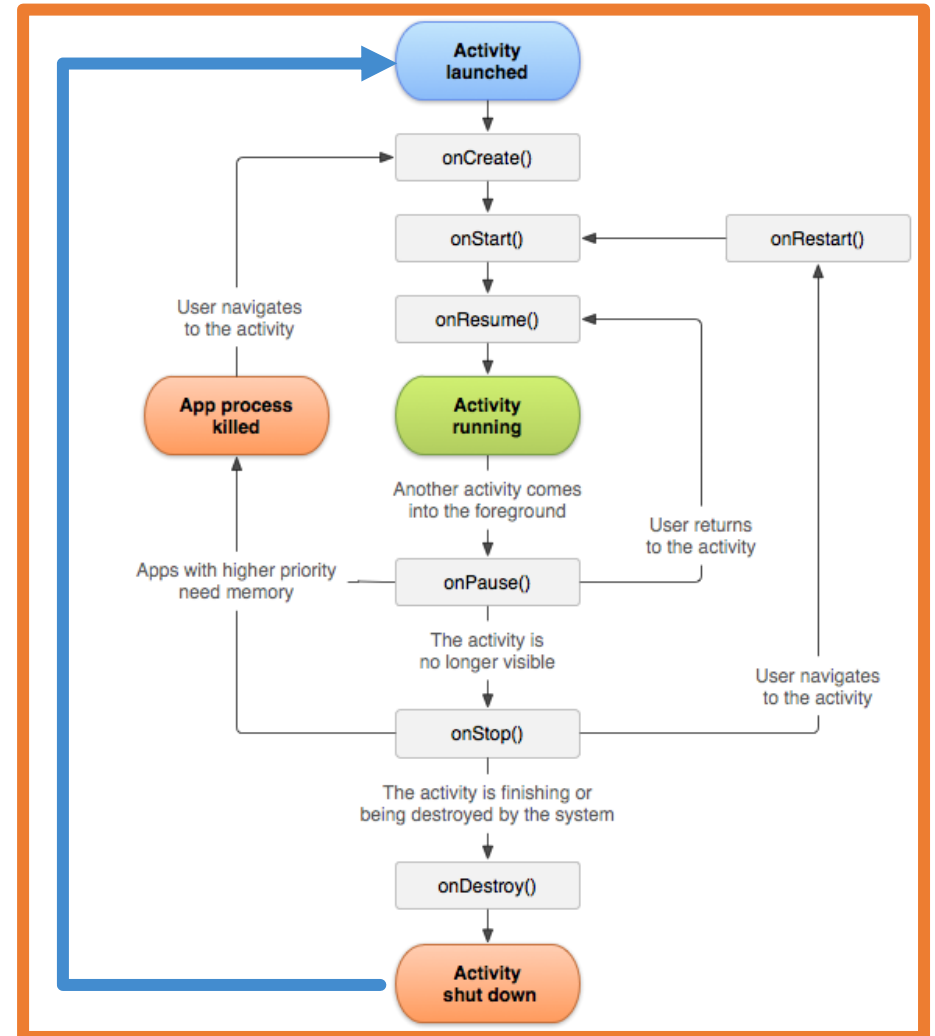
3.- Ciclo de vida de una Activity

Hay que destacar que Android tiene un comportamiento peculiar.

Una Activity activa se destruye y se vuelve a crear cuando:

- Se cambia la orientación del dispositivo.
- Se cambia entre los modos claro y oscuro.
- Se cambia la configuración del dispositivo, por ejemplo el idioma.

Tener en cuenta esto es muy importante a la hora de desarrollar las aplicaciones.



3.- Ciclo de vida de una Activity

La clase **Activity** proporciona una serie de **funciones de retorno** (callbacks) que permiten saber a la actividad que **ha cambiado de estado**:

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onDestroy()

El sistema invoca a cada uno de estos callbacks cuando una actividad cambia de estado de su ciclo de vida (lifecycle).

3.- Ciclo de vida de una Activity

Con los **callbacks** del ciclo de vida se declara cómo se comporta la actividad cuando el usuario deja y vuelve a la actividad.

Por ejemplo:

Si se está creando un reproductor de video por streaming se puede indicar que se pause el vídeo y se finalice la conexión de red si el usuario cambia de aplicación.

Cuando el usuario vuelva a la actividad se puede reconectar y reanudar la reproducción.

4.- Programación imperativa vs programación declarativa

Antes de comenzar a desarrollar aplicaciones Android es necesario comprender dos paradigmas de programación:

- **Programación imperativa:**

Se indica qué se quiere y cómo se quiere hacer.

Es la programación "típica" usando estructuras de control (if, while, for...)

"Hemos visto que hay una mesa libre para dos personas justo al lado de la barra, nos gustaría sentarnos a cenar y primero se sentará mi amigo y luego me sentaré yo."

- **Programación declarativa:**

Se indica qué se quiere.

SQL es un lenguaje declarativo.

"Mesa para dos, por favor"

4.- Programación imperativa vs programación declarativa

El lenguaje de programación **Kotlin** dispone de características que lo acercan a la **programación declarativa**.

Ejemplo: Recorrer un array y transformarlo en otro

- Java: se realiza un for para pasar por todos los elementos, se indica cómo se transforma cada elemento y por último, se almacena en el array final.
- Kotlin: se utiliza la función map en la que se dice que una lista la mapee a otra sin indicar si tiene que crear una lista nueva, ni si tiene que añadir los elementos, ni el orden.

5.- Programación de aplicaciones nativas Android

Hoy en día para desarrollar aplicaciones nativas en Android hay dos opciones:

- Tradicional con **Views** (Vistas): programación imperativa.

La interfaz gráfica se define en archivos XML en los que se indican los elementos gráficos (Views) y en el código del programa se indica cómo se realizan todas las acciones.

- **Jetpack Compose**: programación imperativa-declarativa.

En el código del programa se indican los elementos gráficos (UI declarativas) y qué funcionalidad tienen (programación imperativa).

5.- Programación de aplicaciones nativas Android

Uno de los puntos fuertes de las **interfaces de usuario declarativas** es que los elementos de la interfaz se conectan al **estado de la Activity**.

De esta manera **si un elemento de la interfaz cambia, el estado cambia y la interfaz se repinta** para representar ese nuevo estado.

Esto se realiza **de manera automática** sin tener que indicar nada como programadores.

Este sistema está muy optimizado y **solo las partes afectadas por ese cambio de estado son las que se repintan**.

5.- Programación de aplicaciones nativas Android

Ventajas del uso de interfaces de usuario declarativas:

- Menos código.
- Código más sencillo y fácil de entender/leer.
- Evita clases intermedias que pueden proveer de errores.
- Intuitivas.
- Al engancharse al estado de la aplicación la vista se encarga de todo.
- Muy rápidas.
- Vistas previas de cualquier componente.
- Muy potentes.

Frameworks como **React Native**, **Flutter** o **Swift UI** utilizan el paradigma de interfaces de usuario declarativas.

5.- Programación de aplicaciones nativas Android

En este curso se verá el uso de Jetpack Compose por ser la tendencia actual del mercado.

En el desarrollo de aplicaciones nativas Android se pueden mezclar la manera tradicional con XML y Jetpack Compose.

En Aules están disponibles los apuntes del curso pasado donde se explica el desarrollo de aplicaciones nativas Android de la manera tradicional con Views y archivos XML.

6.- Jetpack Compose

Jetpack Compose es un **kit de herramientas** (toolkit) para crear y compilar **interfaces de usuario declarativas** para Android.

Se basa **100% en Kotlin**.

Se incorpora a partir de la versión **Artic Fox** (2021) de Android Studio.

Las aplicaciones desarrolladas con Jetpack Compose se pueden ejecutar en las versiones de **Android 5.0 (API 21) y superiores**.



6.- Jetpack Compose

Jetpack Compose solo está disponible para Android.

Se compone de:

- **Compilador:** plugin gradle que genera el código necesario.
- **Runtime:** entorno de ejecución que genera y mantiene el árbol de nodos para saber los elementos que se encuentran en la interfaz.
- **Librería de UI:** decide cómo se interpreta y se pinta el árbol de nodos.

6.- Jetpack Compose

Tanto el **compilador** como el **runtime** son "fijos" y trabajan de forma genérica.

La **librería de UI** es un componente que puede cambiar y ahora mismo la única versión estable es Jetpack Compose que es para Android.

Jetbrains está creando las librerías:

- Compose for Desktop (Windows, Mac y Linux)
- Compose for web (experimental)
- Compose for iOS (alpha)

Todas ellas se agrupan junto a Jetpack Compose en el proyecto [Compose Multiplatform](#) que permite desarrollar una aplicación con Compose y generar el ejecutable para Android, iOS, escritorio y web como ya ocurre con Flutter.

7.- Desarrollo Android con Jetpack Compose

El uso de Jetpack Compose para el desarrollo de aplicaciones Android consiste en definir la interfaz gráfica de la aplicación de manera declarativa.

Para definir la interfaz gráfica se usan componentes de Jetpack Compose que pueden ser los ofrecidos por el sistema o bien los propios definidos por el programador.

Un componente Jetpack Compose puede contener a otro componente Jetpack Compose. Es habitual usar este comportamiento para crear componentes propios que extiendan la funcionalidad de otros componentes ya existentes.

Si se necesita también se puede utilizar la programación imperativa: variables, clases, estructuras de control, funciones...

8.- Crear un proyecto Android

La mejor manera de **entender cómo funciona Android Studio** es **realizar una aplicación sencilla** donde se utilicen algunos de los componentes y funcionalidades.

Así, esta unidad va a consistir en explicar los conceptos a la vez que se crea una aplicación.

Algunos conceptos que se verán se explicarán más detenidamente en las siguientes unidades.

La aplicación que realizaremos consistirá en un **contador de clics** como la que se muestra en la imagen.



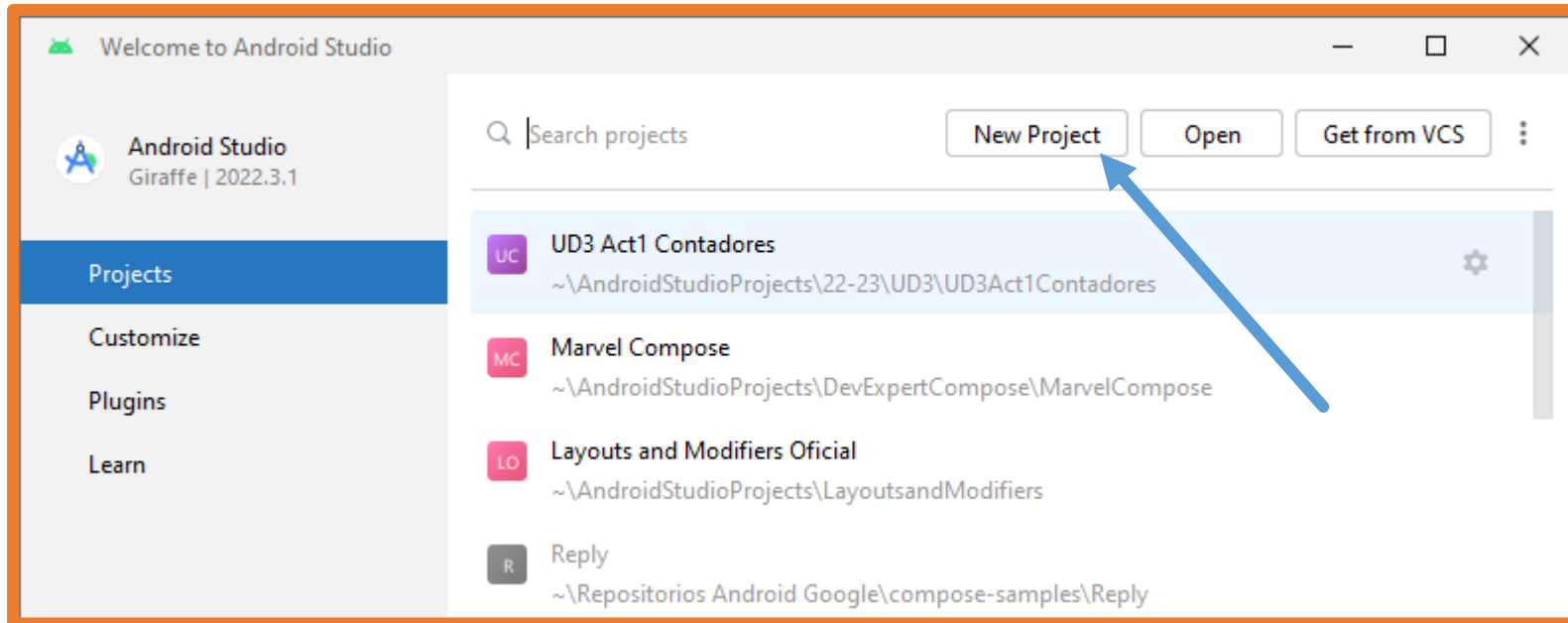
8.- Crear un proyecto Android

Una vez abierto Android Studio hay varias opciones para crear un proyecto:

- Si **no hay abierto** ningún proyecto Android:
Hacer clic en **New Project**.
- Si **hay un proyecto abierto** hay dos opciones:
 - Cerrar proyecto y a continuación hacer clic en **New Project**.
 - Crear un nuevo proyecto directamente.
- Crear proyecto **desde un control de versiones (VCS)**:

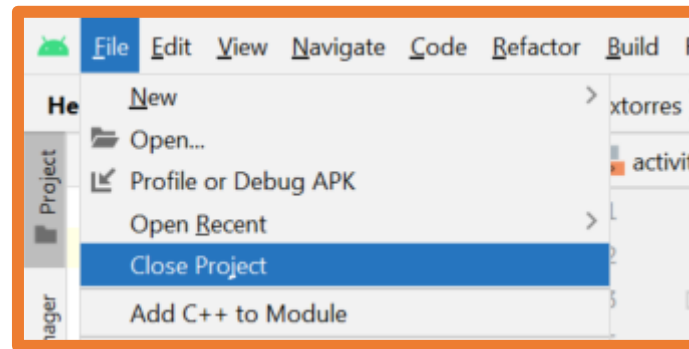
8.- Crear un proyecto Android

- Si **no hay abierto** ningún proyecto Android:
Hacer clic en **New Project**.

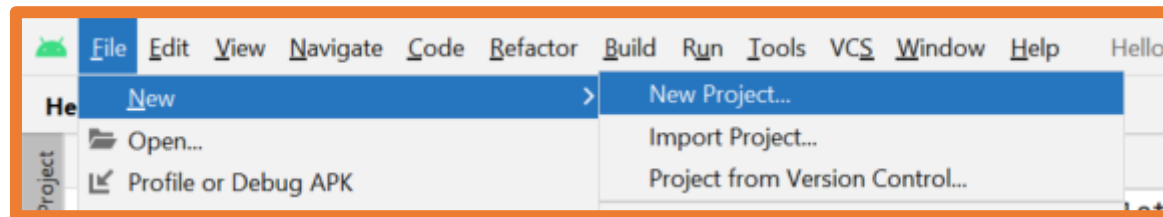


8.- Crear un proyecto Android

- Si **hay un proyecto abierto** hay dos opciones:
 - Cerrar proyecto y a continuación hacer clic en **New Project**.

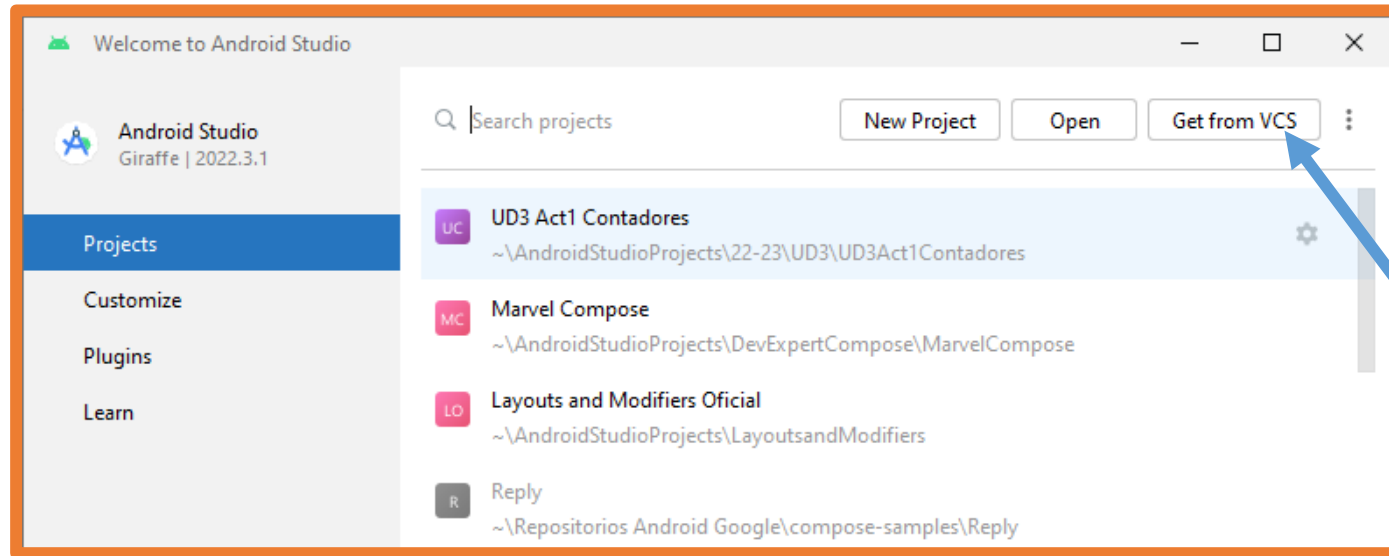


- Crear un nuevo proyecto directamente.



8.- Crear un proyecto Android

- Crear proyecto **desde un control de versiones (VCS)**:
Hacer clic en **Get from VCS**.
Se debe disponer de la URL del repositorio.

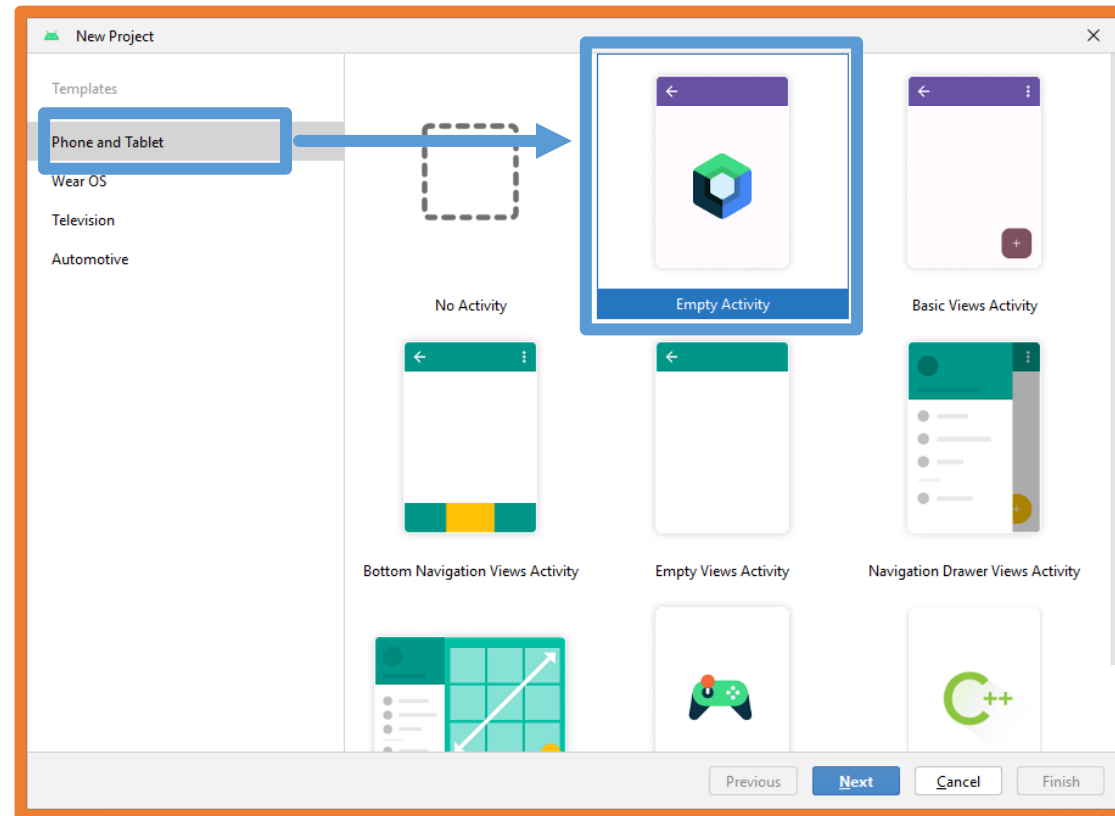


Esta opción descarga un proyecto ya creado y configurado por lo que las siguientes páginas no aplican a esta opción.

8.- Crear un proyecto Android

Al crear el proyecto, Android Studio muestra una ventana con todas las plantillas disponibles.

Se debe elegir **Phone and Tablet** y a continuación **Empty Activity** (utiliza Jetpack Compose).



Las plantillas con el texto Views en el nombre utilizan la programación tradicional con XML.

8.- Crear un proyecto Android

A continuación, se debe rellenar las opciones del proyecto:

- **Nombre:** debe ser significativo.
- **Paquete:** debe ser único, para estar seguros de esto se seguirá la siguiente estructura
com.tunombretuapellido.nombreproyecto
- **Save location:** directorio que se quiera
- **Minimum SDK:** API 24 ("Nougat"; Android 7.0)

New Project

Empty Activity

Create a new empty activity with Jetpack Compose

Name: Contador de clics

Package name: com.alextores.contadordeclics

Save location: C:\Users\Alex\AndroidStudioProjects\23-24\UD3\Contadordeclics

Minimum SDK: API 24 ("Nougat"; Android 7.0)

ⓘ Your app will run on approximately 95,4% of devices.
[Help me choose](#)

Build configuration language ⓘ Kotlin DSL (build.gradle.kts) [Recommended]

Previous Next Cancel Finish

8.- Crear un proyecto Android

SDK mínimo

La elección del SDK mínimo es un paso crucial en el inicio de un proyecto.

- Versión más baja posible → soporte a la mayor cantidad de dispositivos.
- Versión más alta posible → tener todas las características y funcionalidades.

En **todas las actividades del curso** se va a elegir la versión **API 24** (alcance de un 95,4%).


Aunque se podría elegir sin problema las versiones **API 27** (90,2%) o **API 28** (84,1%) ya que hoy en día pocos dispositivos están por debajo de esas versiones.

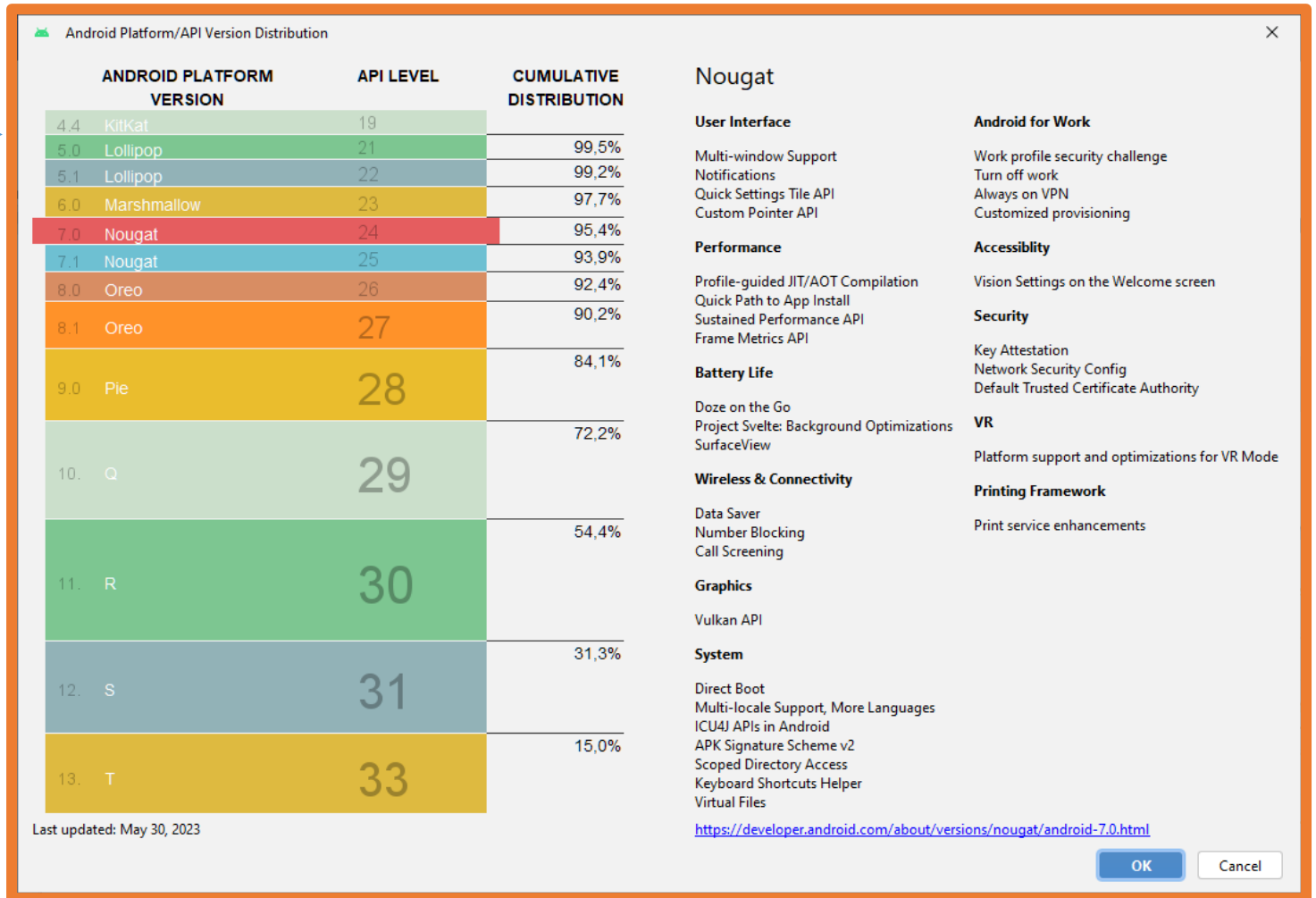
8.- Crear un proyecto Android

SDK mínimo

Minimum SDK

API 24 ("Nougat"; Android 7.0)

 Your app will run on approximately 95,4% of devices.
[Help me choose](#)



8.- Crear un proyecto Android

Una vez seleccionadas todas las opciones se debe hacer clic en **Finish**.

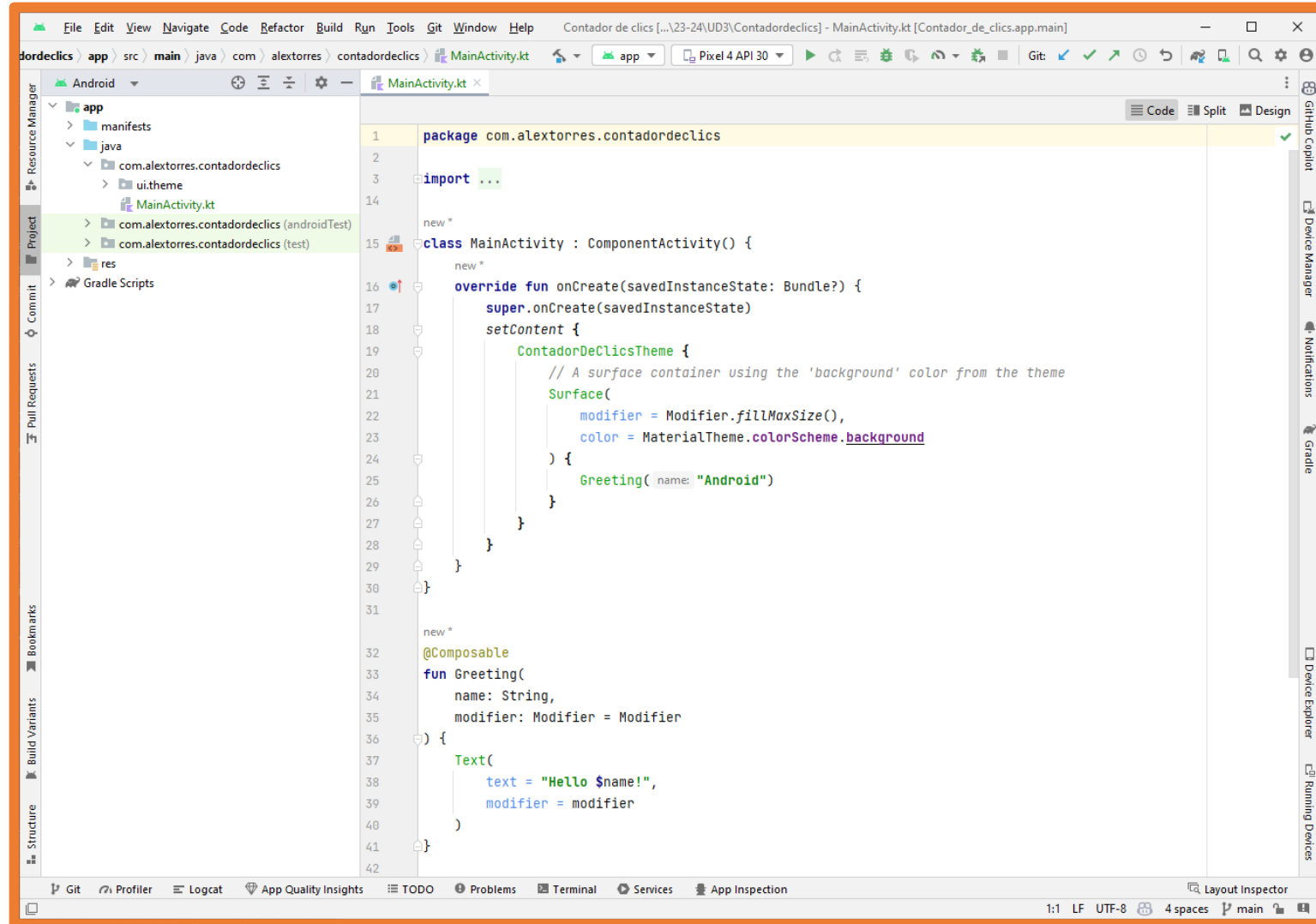
En ese punto **Android Studio comenzará a crear el proyecto**.

Durante la creación del proyecto Android Studio realizará la descarga de todos los componentes necesarios.

Se recomienda no interactuar con el programa hasta que no finalice por completo la creación del proyecto.

En la parte inferior derecha se puede consultar la barra de progreso con las descargas y creación del proyecto.

8.- Crear un proyecto Android

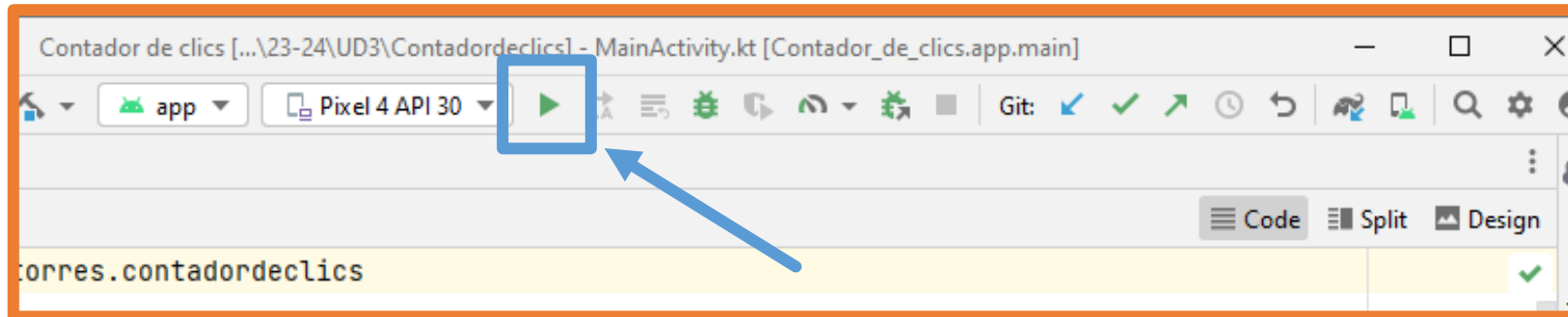


Práctica

Actividad

Crear proyecto llamado "Test".

Ejecútalo en el emulador de Android Studio.

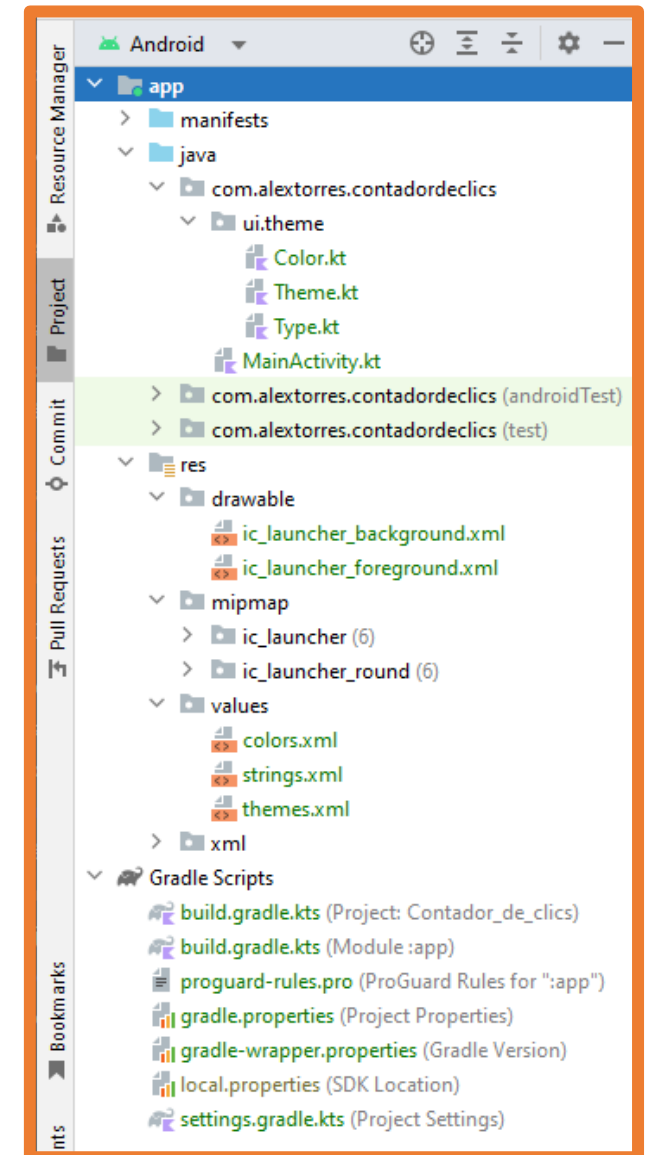


9.- Estructura de un proyecto Android

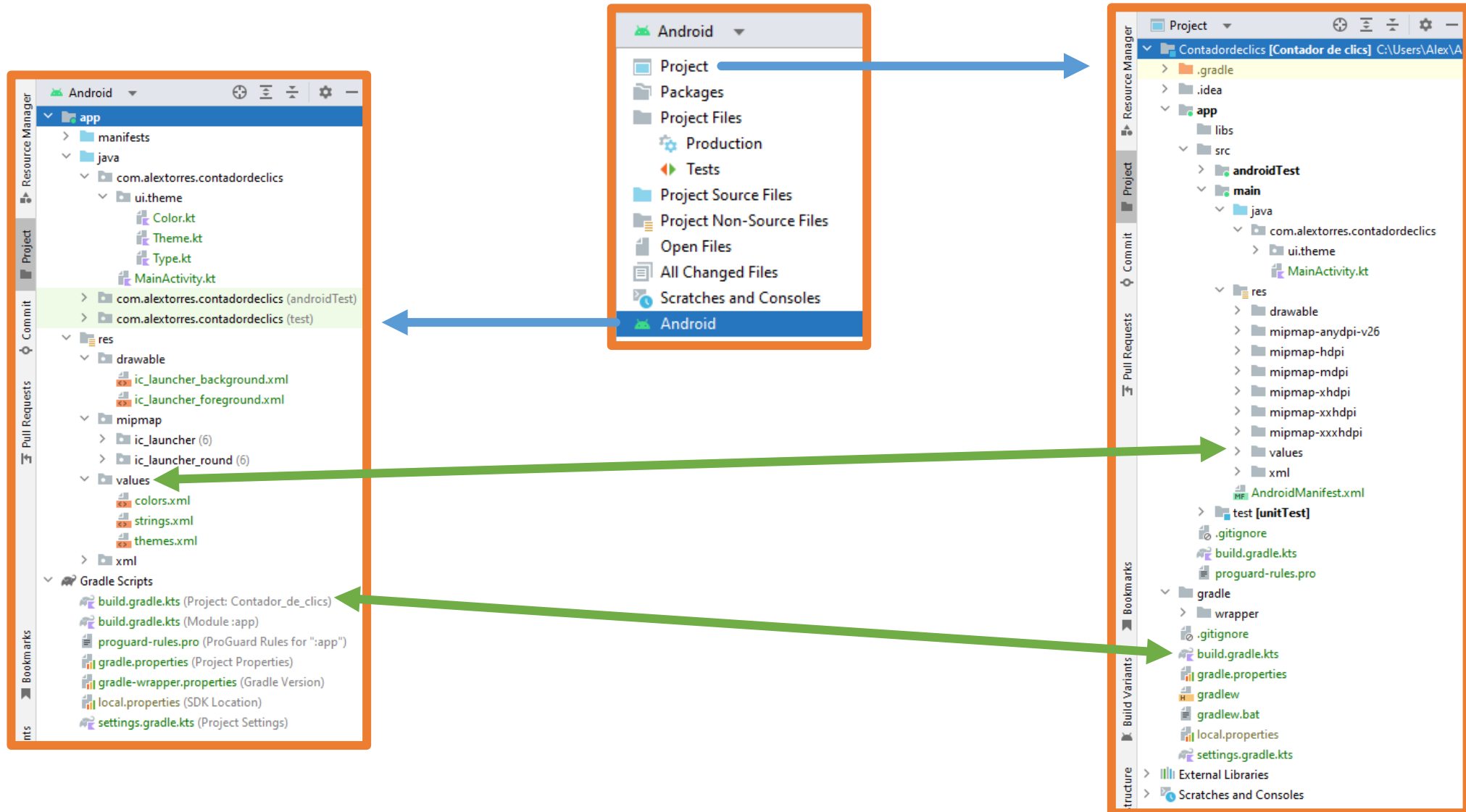
Una vez creado el proyecto se pueden ver todos los **archivos del mismo**.

Depende de la visualización que se seleccione (Project, Android...) los archivos se podrán encontrar en un lugar o en otro.

A continuación, se explicará los más importantes.



9.- Estructura de un proyecto Android



9.- Estructura de un proyecto Android

Archivo: AndroidManifest.xml

Project → app/src/main/AndroidManifest.xml

Android → manifests/AndroidManifest.xml

Describe las características fundamentales de la aplicación y sus componentes.

En él se especifican las **Activities** (pantallas) que tiene la aplicación y qué permisos requieren dichas Activities: cámara, contactos, internet...

9.- Estructura de un proyecto Android

Archivos: build.gradle

Android Studio utiliza **Gradle** para compilar y construir la aplicación.

Hay un archivo **build.gradle** para todo el proyecto y otro archivo **build.gradle** por cada módulo del proyecto.

Por lo general, solo interesará el archivo **build.gradle** del módulo **app**.

En este archivo están las dependencias de compilación de la aplicación y también la configuración predeterminada.

Project → app/build.gradle

Android → Gradle Scripts/build.gradle.kts (Module: app)

9.- Estructura de un proyecto Android

Archivo: build.gradle.kts (Module: app)

- compiledSdk → Versión a la que se va a compilar.
Por defecto, es la última versión SDK instalada en el ordenador
- applicationId → Nombre completo del paquete de la aplicación.
- minSdk → Versión mínima de SDK especificada al crear el proyecto.
Será la versión más antigua que admita la aplicación.
- targetSdk → Versión más alta con la que se prueba la aplicación.
- dependencias → Sección donde añadir las dependencias que se quieren instalar para la aplicación.

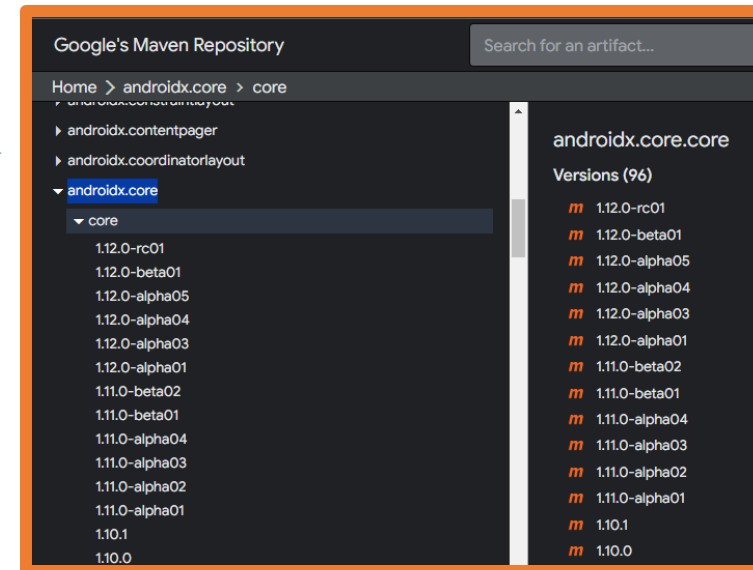
9.- Estructura de un proyecto Android

Archivo: build.gradle.kts (Module: app)

dependencias → Sección donde añadir las dependencias que se quieren instalar para la aplicación.

Si se quieren consultar las últimas versiones de las dependencias ofrecidas por Google para Android se debe ir a <https://maven.google.com/web/index.html>.

```
implementation("androidx.core:core-ktx:1.10.1")  
implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.1")  
implementation("androidx.activity:activity-compose:1.7.2")
```



Para otras dependencias se debe consultar su documentación oficial.

9.- Estructura de un proyecto Android

Archivo: MainActivity.kt

Project → app/src/main/java/com.alextores.contadordeclics/MainActivity.kt

Android → java/com.alextores.contadordeclics/MainActivity.kt

En este archivo se **programará el comportamiento de esta ventana.**

Por defecto contiene:

- La definición de la Activity principal y su método onCreate. Por el ciclo de vida de las Activities el código de onCreate se ejecutará cuando la actividad alcance ese estado.
- Componente de Jetpack Compose con un texto (Text).
- Componente de Jetpack con una previsualización (@Preview).

9.- Estructura de un proyecto Android

Carpeta: ui/theme

Project → app/src/main/java/com.alextores.contadordeclics/ui.theme

Android → java/com.alextores.contadordeclics/ui.theme

En este directorio se encuentran los archivos que permiten configurar el tema que usa la aplicación.

Por defecto Jetpack Compose utiliza un tema basado en [Material Design](#) (guía de diseño de interfaces diseñada por Google).

Con los archivos incluidos en ui.theme se puede extender ese tema.

9.- Estructura de un proyecto Android

Carpeta: res

Project → app/src/main/res

Android → res

Este directorio contiene los recursos de la aplicación.

Carpeta: drawable

Project → app/src/main/res/drawable

Android → res/drawable

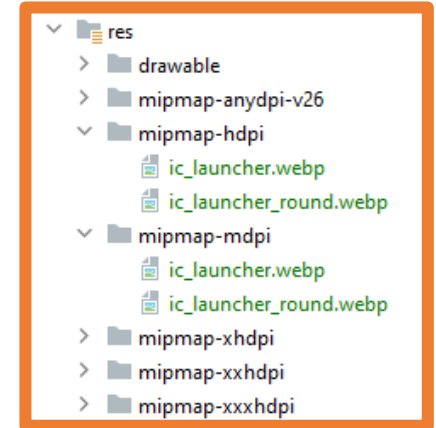
Directorio donde almacenar las imágenes de la aplicación

9.- Estructura de un proyecto Android

Carpeta: mipmap

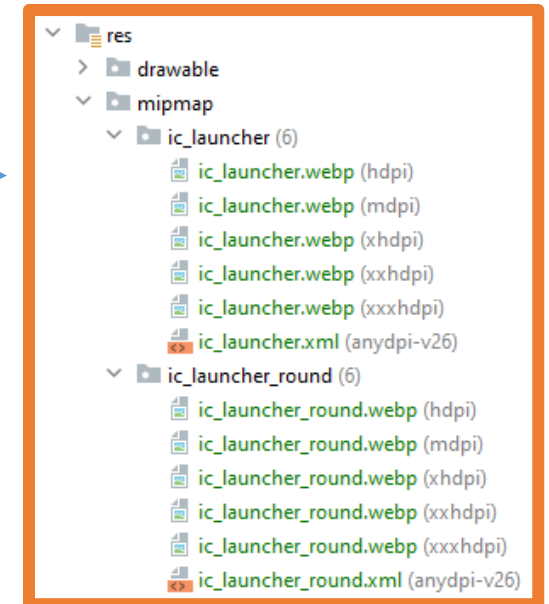
Project → app/src/main/res/mipmap-*RESOLUCIÓN*

Directorios que contienen el icono de la aplicación para las diferentes densidades de píxeles de pantalla.



Android → res/mipmap

En la vista Android se agrupan los archivos por su nombre. Junto al nombre se puede ver la *RESOLUCIÓN* en la que se utilizan.



9.- Estructura de un proyecto Android

Carpeta: mipmap

Nomenclatura de *RESOLUCIÓN* en Android:

- xxxhdpi → 640 dpi
- xxhdpi → 480 dpi
- xhdpi → 320 dpi
- hdpi → 240 dpi
- mdpi → 160 dpi

10.- Configuración de Android Studio

Jetpack Compose está integrado totalmente dentro de Android Studio.

Aún así hay algunas configuraciones que se puede cambiar para mejorar la productividad y la experiencia del programador.

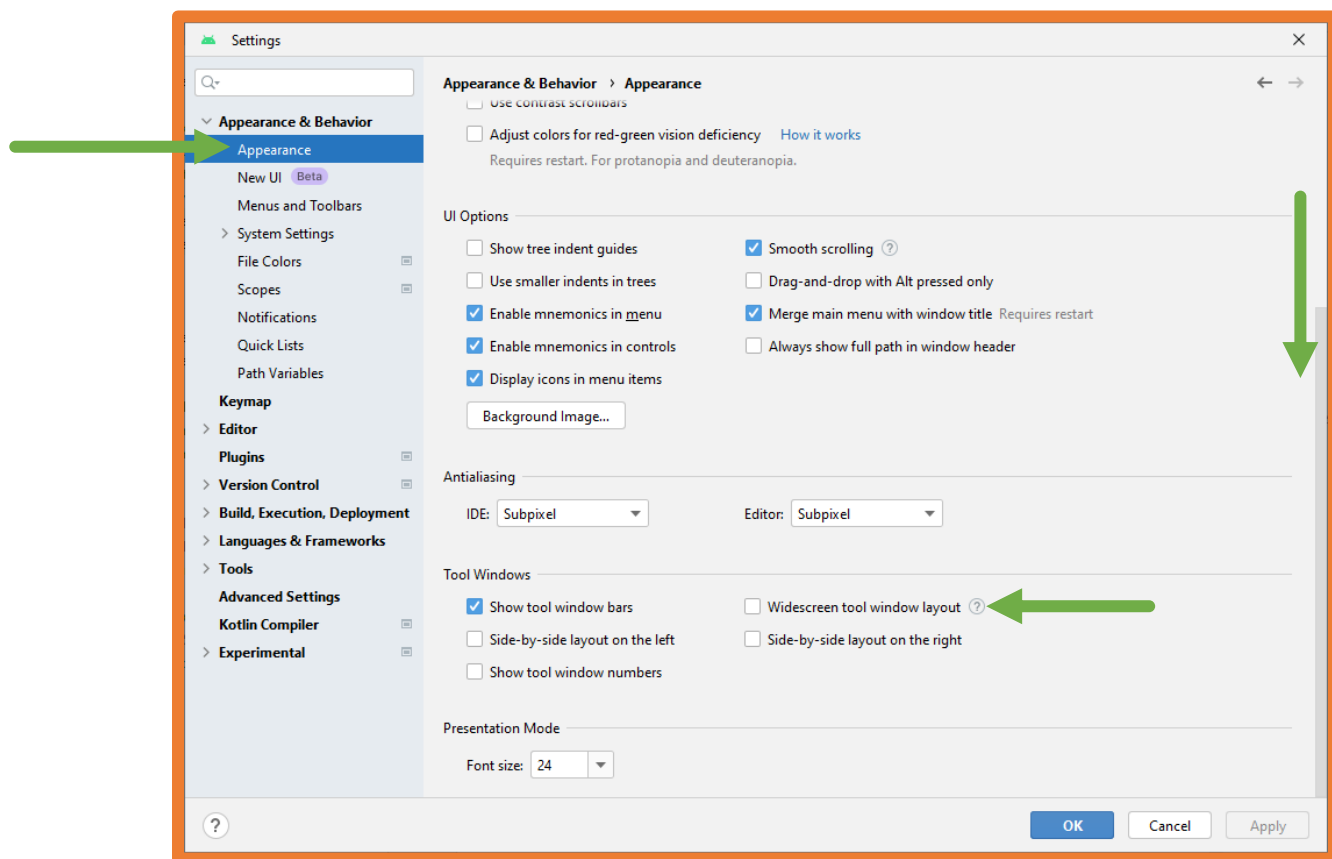
A continuación, se muestran algunas de estas configuraciones que pueden ser de ayuda durante el curso.

File → Settings **CTRL+ALT+S**

10.- Configuración de Android Studio

Barras laterales siempre visibles

Si se dispone de una pantalla ultra ancha es posible que siempre se quieran visibles las barras laterales que generalmente muestran: Project (árbol de archivos del proyecto) y Running Devices (Emulador).



10.- Configuración de Android Studio

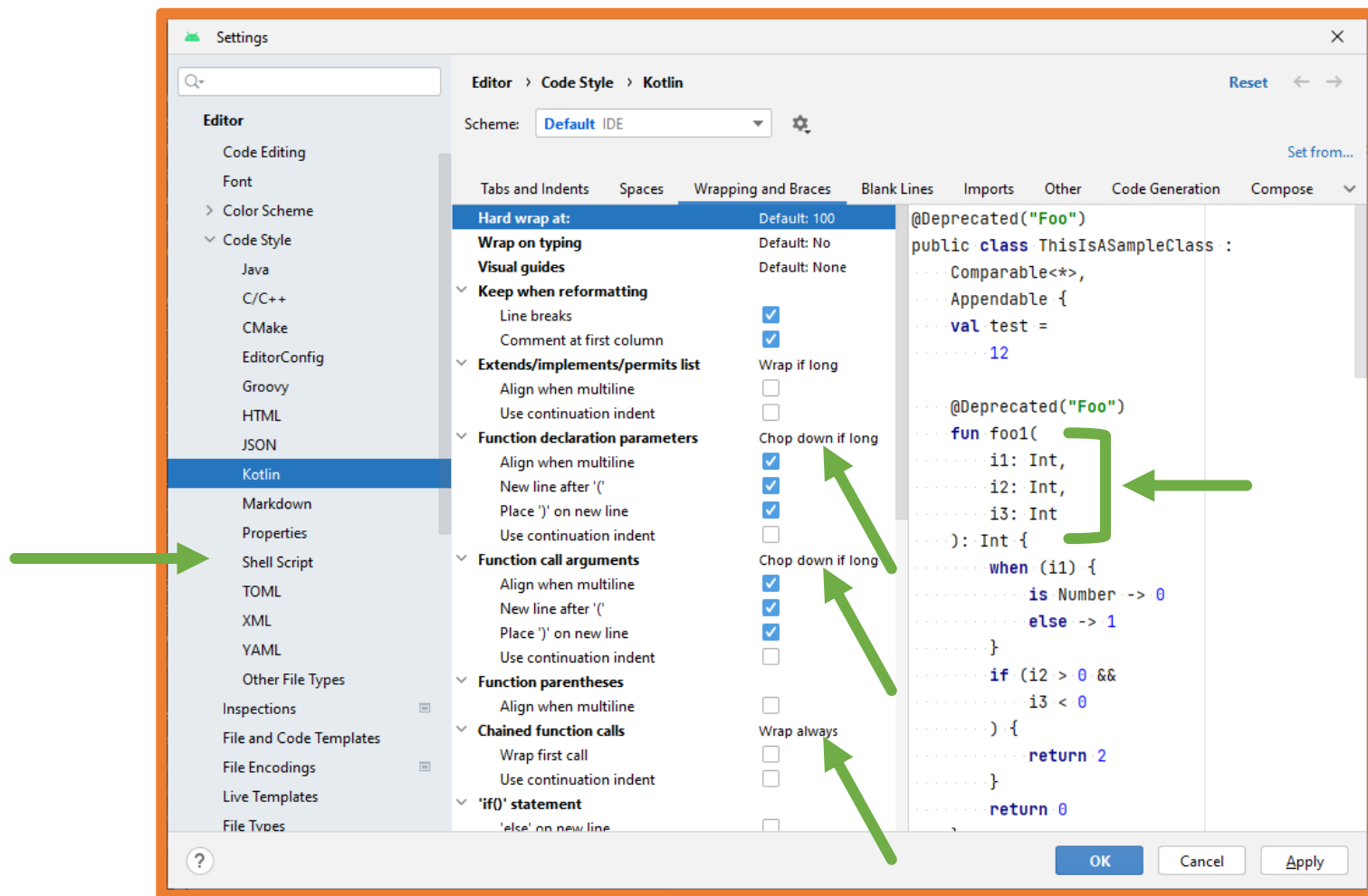
Formatear el código con las guías de estilo de Jetpack Compose

IntelliJ IDEA y **Android Studio** están desarrollados por **JetBrains** por lo que la combinación de teclas **CTRL+ALT+L** usada anteriormente formatea el código automáticamente con las guías de estilo de Kotlin.

Aún así, para visualizar mejor el código Kotlin para Jetpack Compose se pueden cambiar algunas opciones.

10.- Configuración de Android Studio

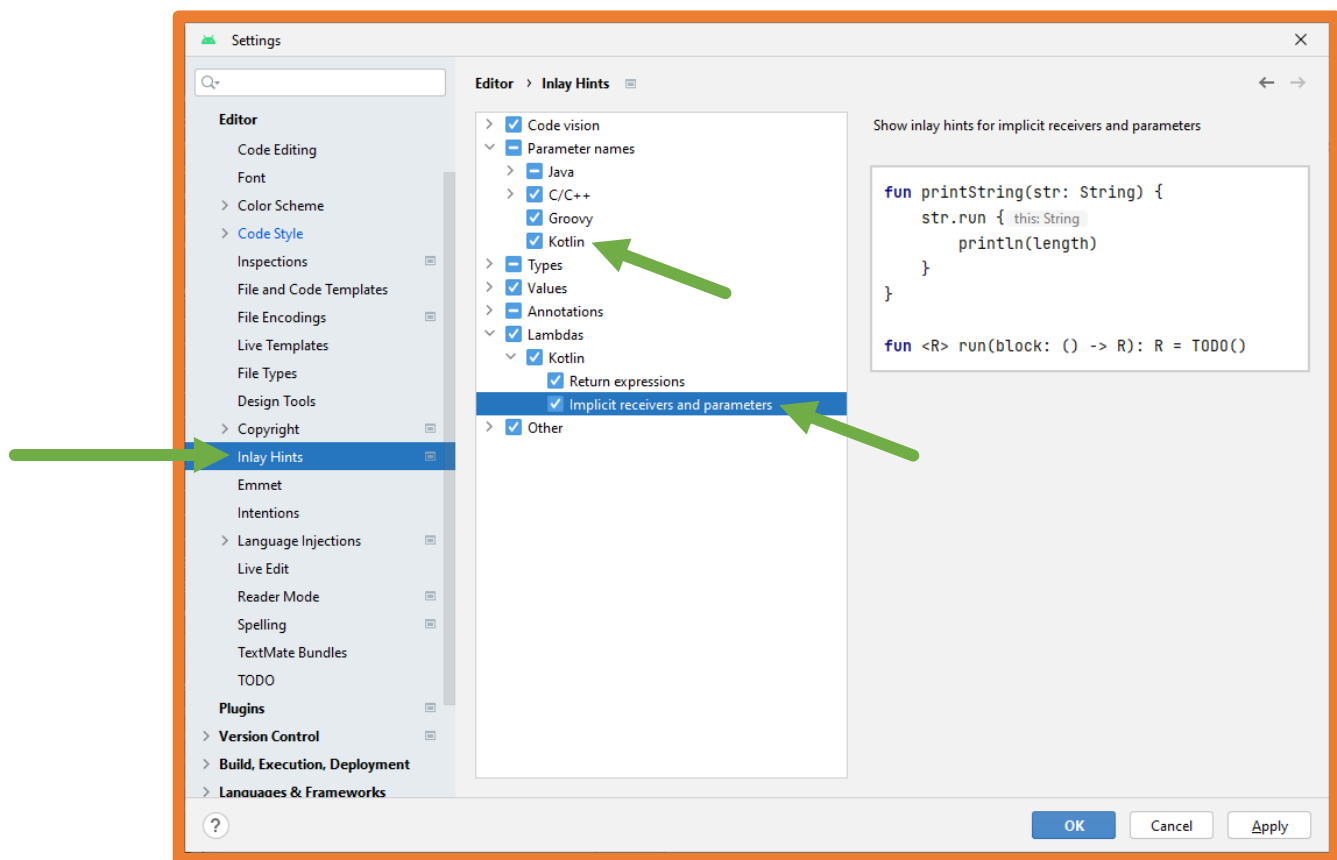
Formatear el código con las guías de estilo de Jetpack Compose



10.- Configuración de Android Studio

Pistas de código

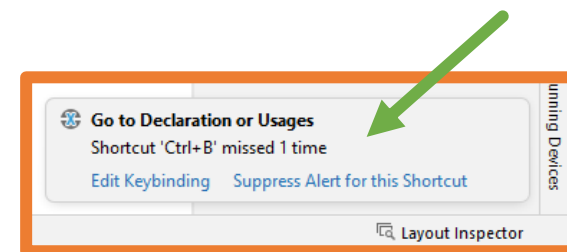
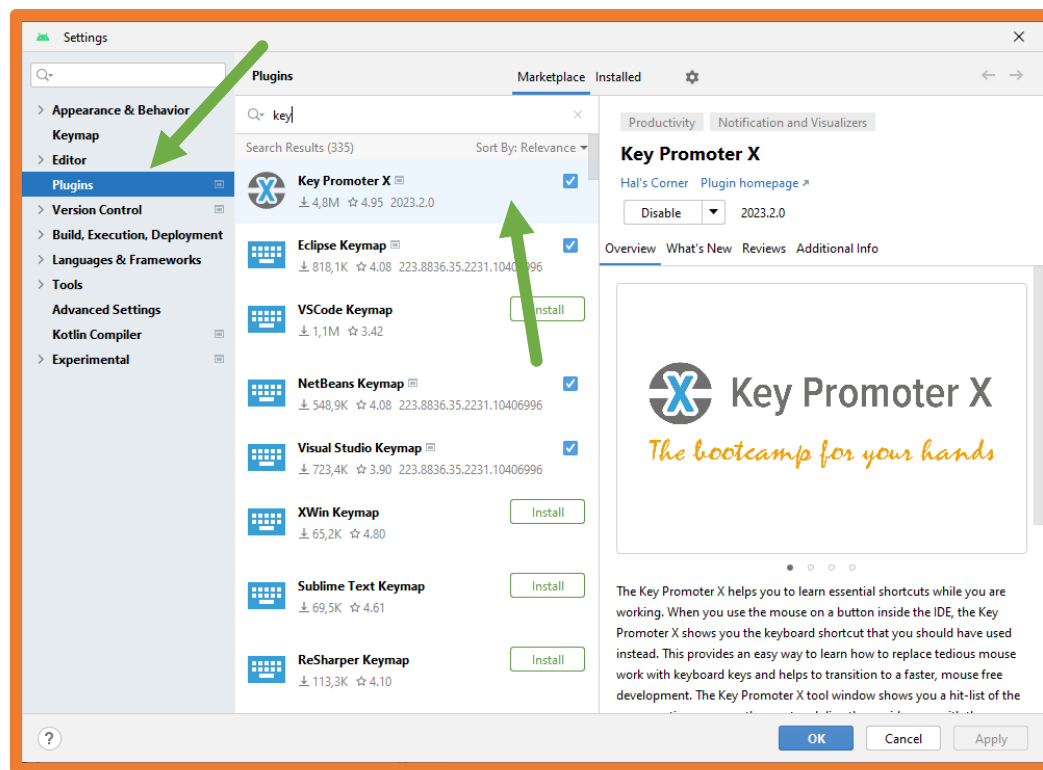
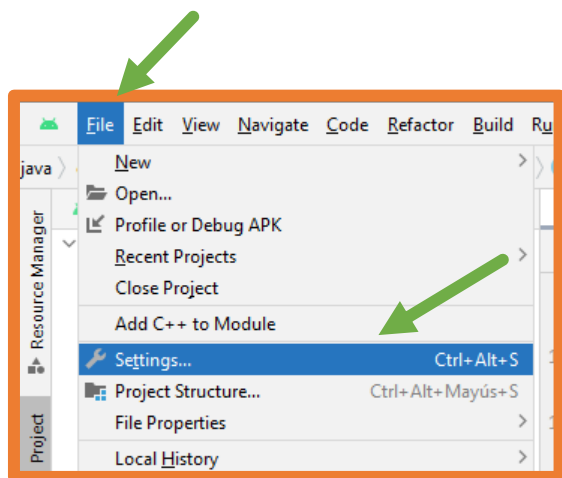
Android Studio puede mostrar en el editor diferentes pistas en el código que ayudan al programador a conocer el orden de los parámetros en las llamadas a las funciones, contexto en el que se está...



10.- Configuración de Android Studio

Plugins

Android Studio dispone de los mismos plugins que IntelliJ IDEA.



11.- Unidades de medida en Android

A la hora de desarrollar aplicaciones Android es muy importante conocer las unidades de medida que se utilizan.

Android puede utilizar las siguientes unidades de medida:

- dp → **d**ensity-independent **p**ixels
- sp → **s**cale-independent **p**ixels
- in → pulgadas
- mm → milímetros
- pt → puntos
- px → píxeles.

11.- Unidades de medida en Android

- **dp (densiti-independent pixels):**

1pd equivale a un píxel en una pantalla de 160dpi.

Es una **unidad flexible que cambiará según los dpi de la pantalla:**

$$dp = (\text{ancho en píxeles} * 160) / \text{densidad de la pantalla}$$

Es la solución más eficiente para mostrar elementos de manera uniforme en pantallas con diferentes densidades.

Se usa para todos los tamaños/medidas/distancias menos las del texto.

- **sp (scale-independent pixels):**

Unidad similar a dp pero que **se escala según el tamaño de fuente.**

Se ajusta a la densidad de pantalla y a las **preferencias del usuario en el sistema.**

Se usa para texto.

11.- Unidades de medida en Android

- **in** (pulgadas):
Pulgadas reales según el tamaño físico de la pantalla.
- **mm** (milímetros)
Milímetros reales según el tamaño físico de la pantalla.
- **pt** (puntos)
Un punto es $1/72$ de una pulgada según el tamaño físico de la pantalla.
- **px** (píxeles):
Se corresponde con un píxel **real** de la pantalla.
No se aconseja su uso debido a que los diferentes dispositivos tienen diferentes densidades de píxeles → **ppi** (pixels per inch).