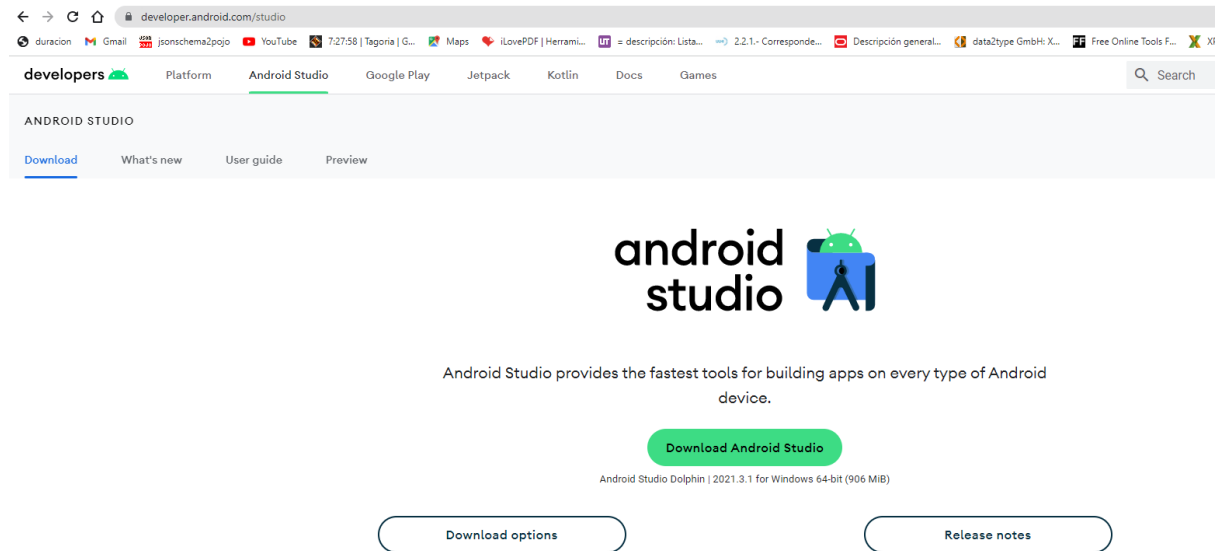


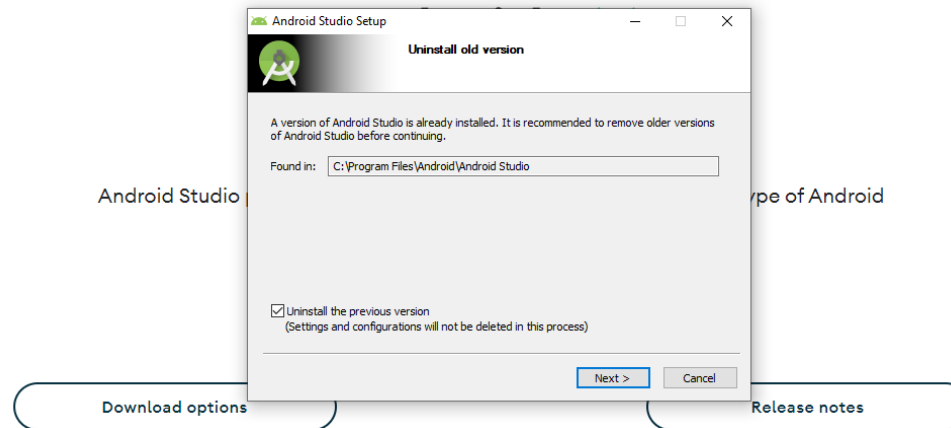
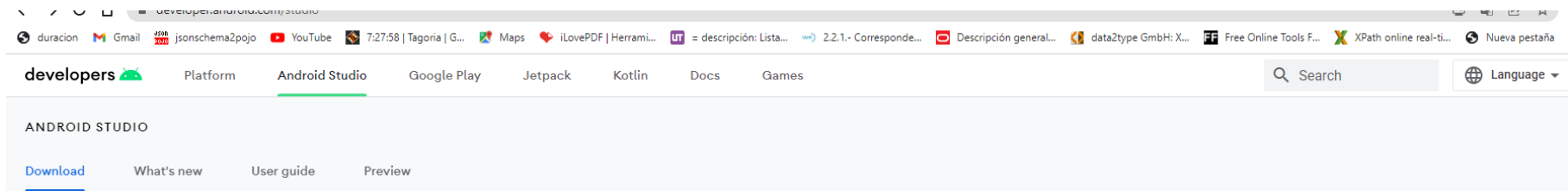
Descargar e instalar Android Studio

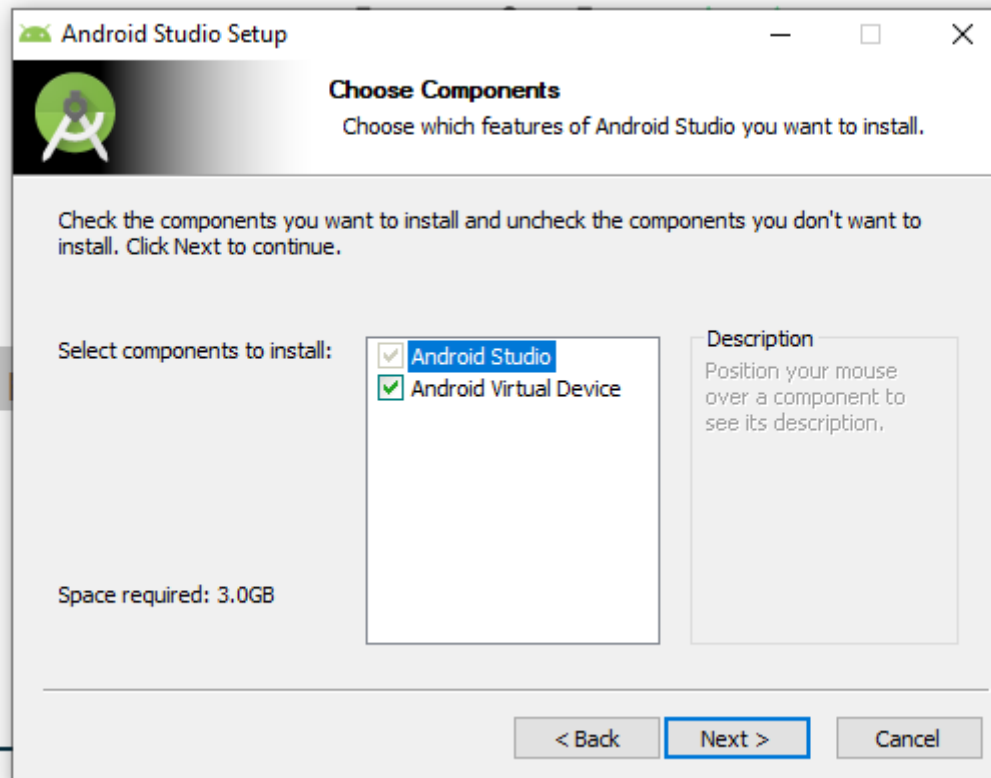
Página de descarga del IDE



Pulsamos en Download Android Studio y aceptamos los términos y condiciones. La descarga comienza automáticamente. Haciendo click en el ejecutable, escogemos la ruta en la que queremos que se descargue y comienza la instalación.

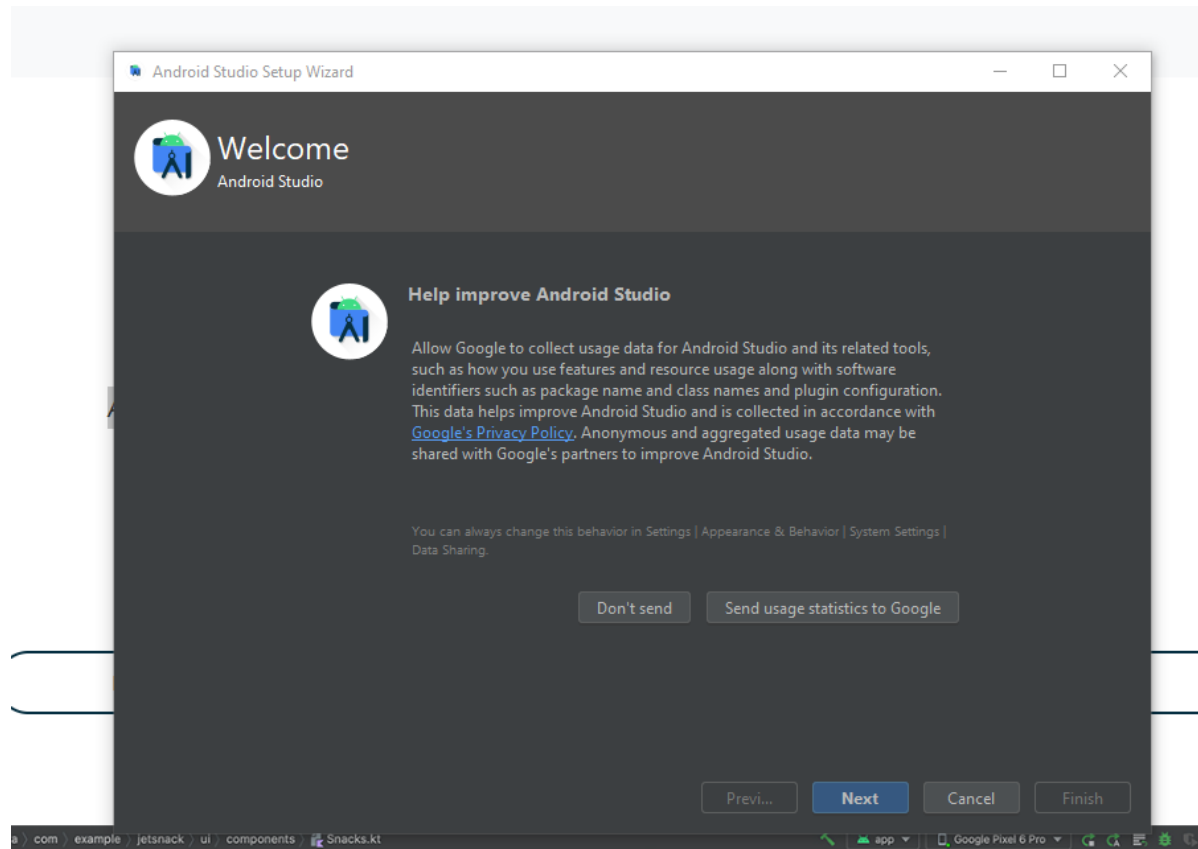






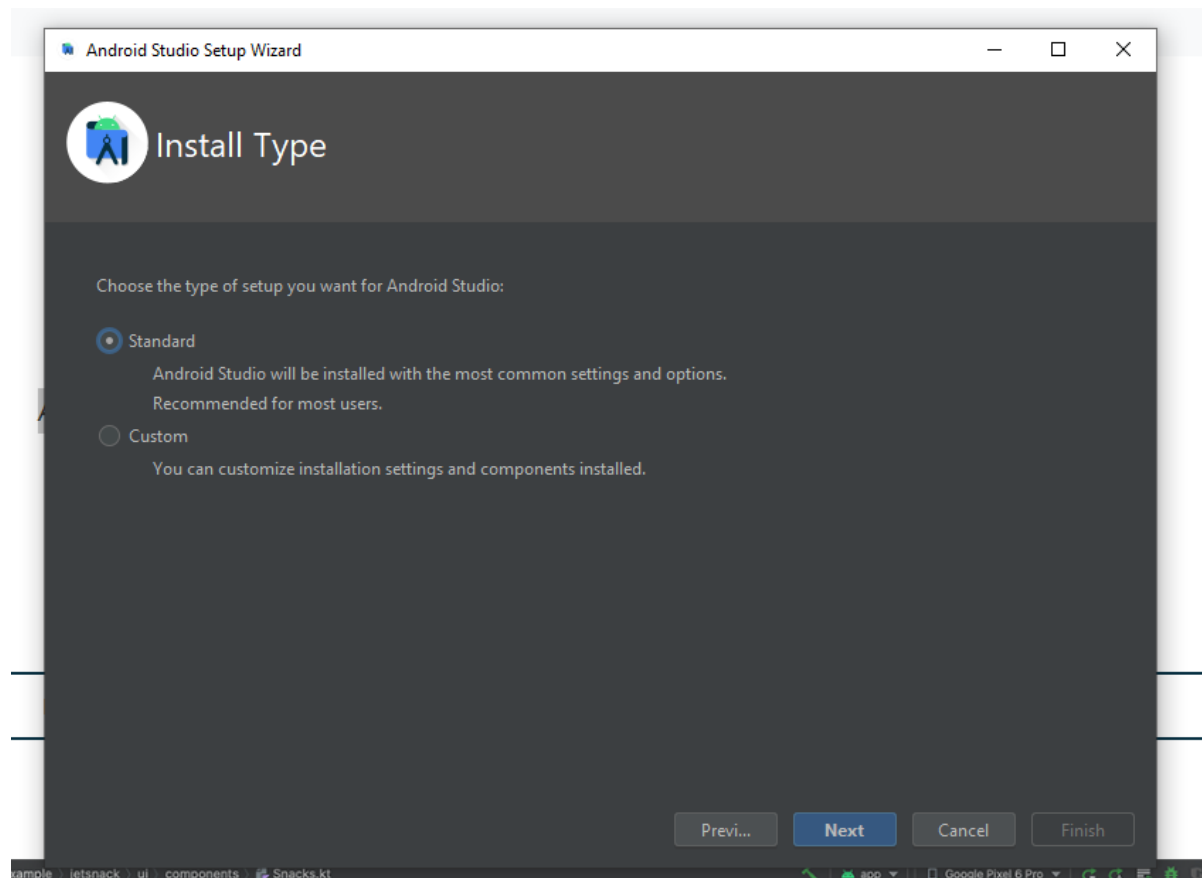
Escogemos los componentes que queremos instalar y nos informa del espacio requerido.



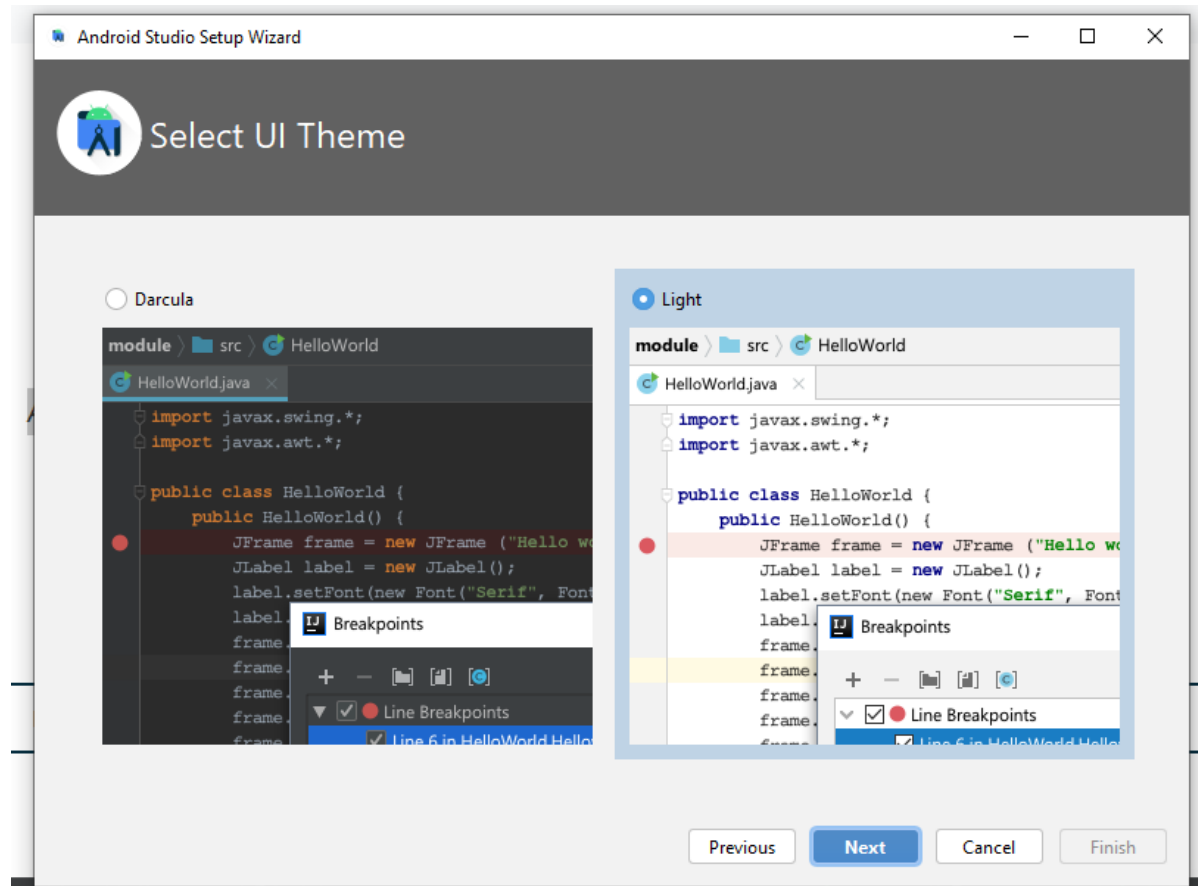


En este momento nos da la opción de hacer una instalación personalizada o de escoger la instalación por defecto.





También nos permite escoger el tema (claro, oscuro)durante la instalación.



Verificamos las especificaciones, SDK y demás.

SDK es el acrónimo de “Software Development Kit” (Kit de desarrollo de software). El SDK reúne un grupo de herramientas que permiten la programación de aplicaciones móviles.

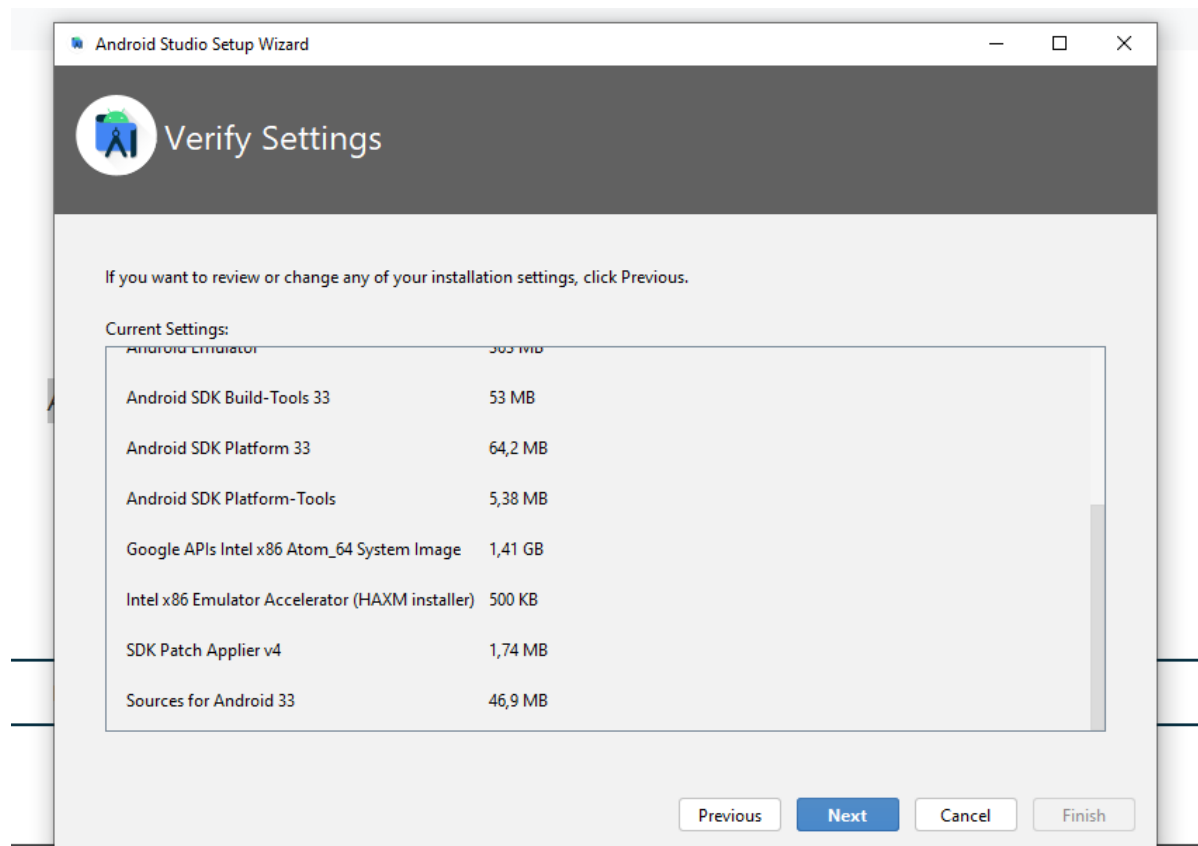
El instalador nos ofrece por defecto la versión más nueva de la SDK, pero habrá que pensarse cuál instalar porque en función del nivel de API, será funcional o no en determinados dispositivos.

En la tabla que adjunto a continuación, hay estimaciones del porcentaje de dispositivos que podrían utilizar cada aplicación en función de la SDK con la que esté creada.



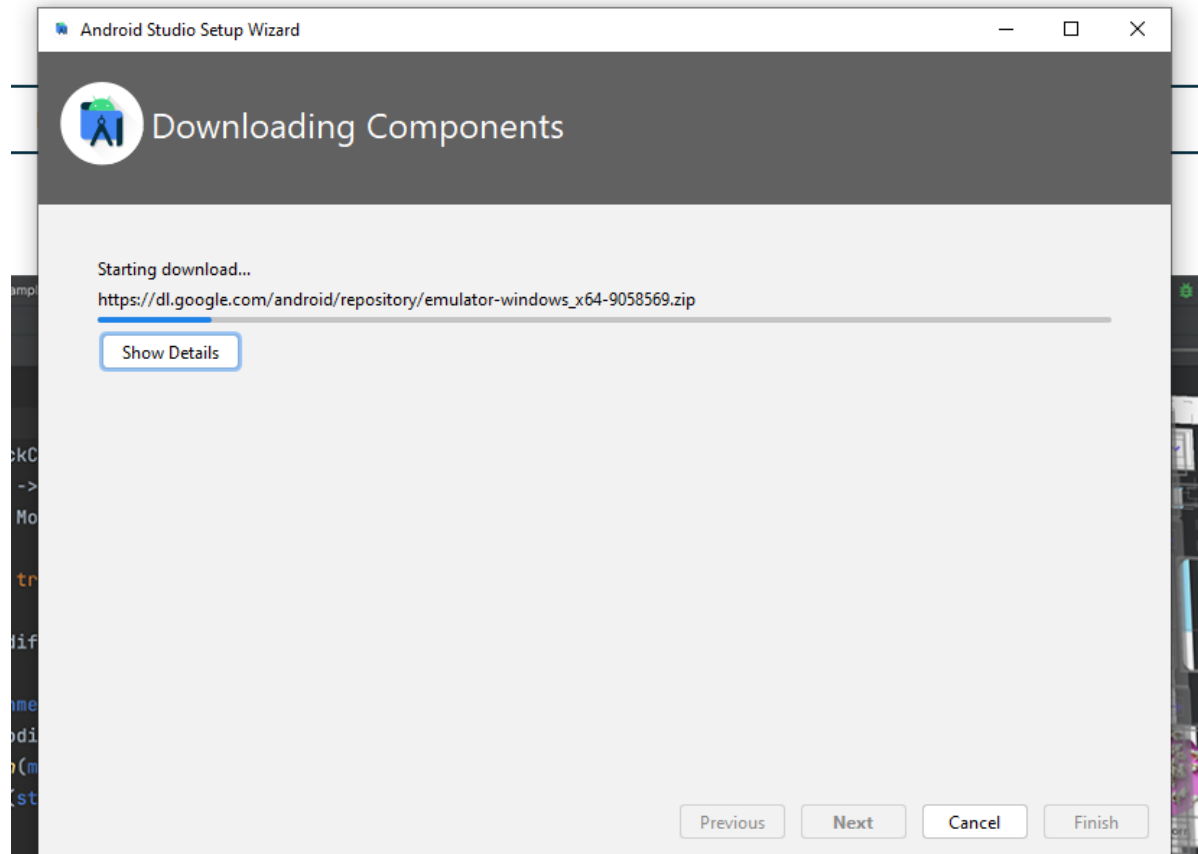
Version	SDK / API level	Version code	Codename	Cumulative usage ¹	Year
Android 13	Level 33	TIRAMISU	Tiramisu ²	No data	2022
Android 12	Level 32 Android 12L	S_V2	Snow Cone ²	<div></div> 20.7%	2021
	Level 31 Android 12	S			
	<ul style="list-style-type: none">▪ targetsdk must be 31+ for new apps.▪ targetsdk will need to be 31+ for app updates by Nov 2022 and all existing apps by Nov 2023. ³				
Android 11	Level 30	R	Red Velvet Cake ²	<div></div> 50.3%	2020
	<ul style="list-style-type: none">▪ targetsdk must be 30+ for app updates, and new WearOS apps.▪ targetsdk will need to be 30+ for all existing apps by November 2022. ³				
Android 10	Level 29	Q	Quince Tart ²	<div></div> 72.1%	2019
Android 9	Level 28	P	Pie	<div></div> 82.9%	2018
	<ul style="list-style-type: none">▪ targetsdk must be 28+ for Wear OS app updates.				
Android 8	Level 27 Android 8.1	O_MR1	Oreo	<div></div> 88.4%	2017
	Level 26 Android 8.0	O		<div></div> 91.1%	
Android 7	Level 25 Android 7.1	N_MR1	Nougat	<div></div> 92.5%	2016
	Level 24 Android 7.0	N		<div></div> 95.1%	
Android 6	Level 23	M	Marshmallow	<div></div> 97.4%	2015
Android 5	Level 22 Android 5.1	LOLLIPOP_MR1	Lollipop	<div></div> 98.8%	2015
	Level 21 Android 5.0	LOLLIPOP, L		No data	2014
	<ul style="list-style-type: none">▪ Jetpack Compose requires a minSdk of 21 or higher.				
Android 4	Level 20 Android 4.4W ⁴	KITKAT_WATCH	KitKat		2013
	Level 19 Android 4.4	KITKAT			
	<ul style="list-style-type: none">▪ Google Play services do not support Android versions below API level 19.				
	Level 18 Android 4.3	JELLY_BEAN_MR2	Jelly Bean		2012
	Level 17 Android 4.2	JELLY_BEAN_MR1			
	Level 16 Android 4.1	JELLY_BEAN			
	Level 15 Android 4.0.3 – 4.0.4	ICE_CREAM_SANDWICH_MR1	Ice Cream Sandwich		2011
	Level 14 Android 4.0.1 – 4.0.2	ICE_CREAM_SANDWICH			





Leemos y aceptamos todos los términos y comienza la instalación de componentes



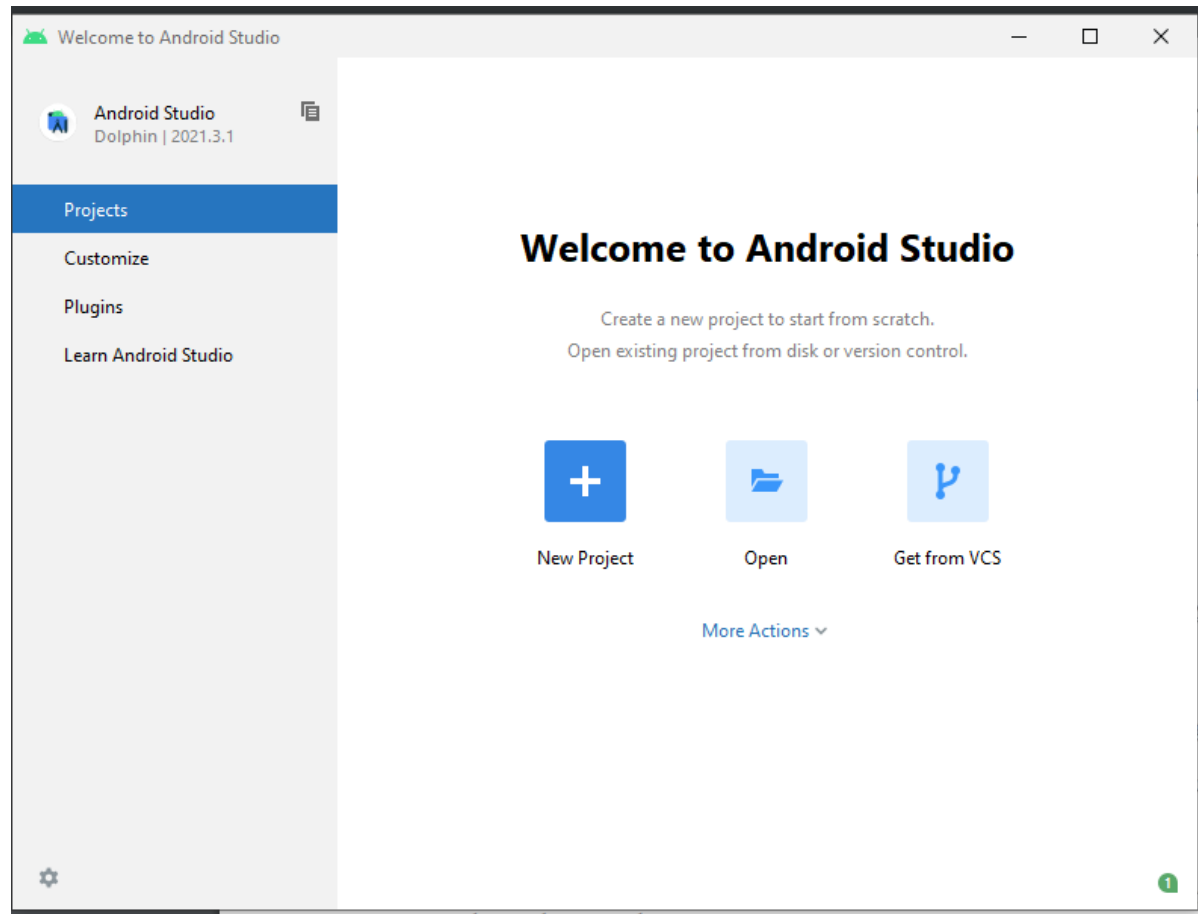


Esta fase de la instalación lleva un tiempo.

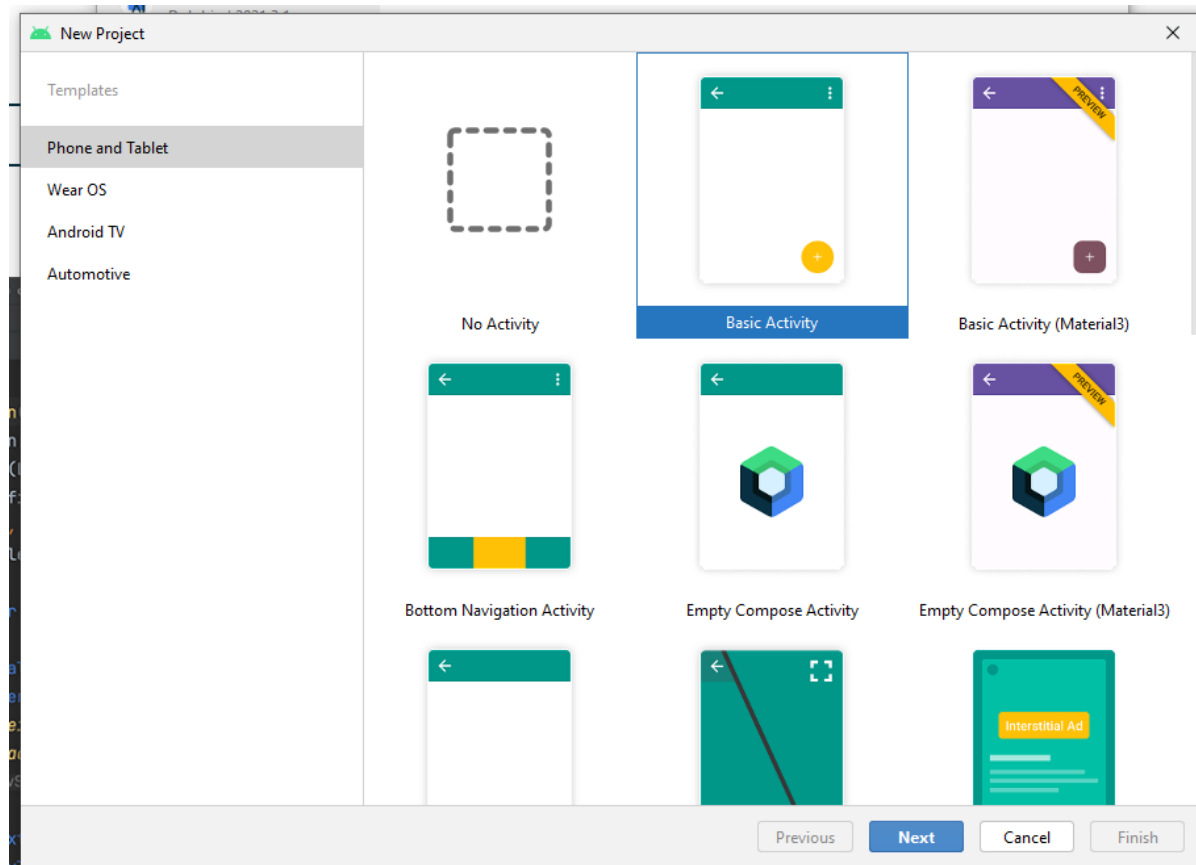


Crear un proyecto con Kotlin

Vamos a crear un nuevo proyecto en Kotlin pulsando en Projects y New Project.

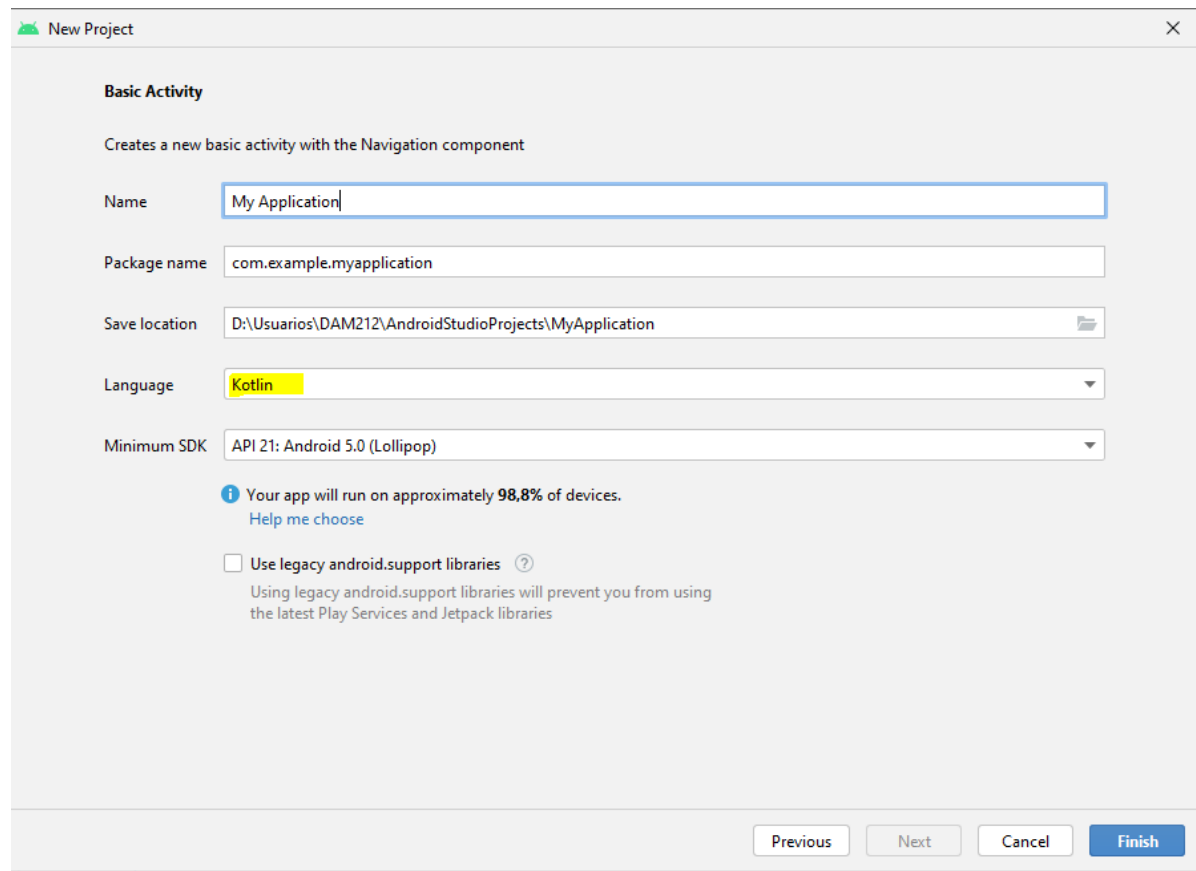


En este caso seleccionamos una basic Activity, aunque también podríamos haber escogido una empty activity.



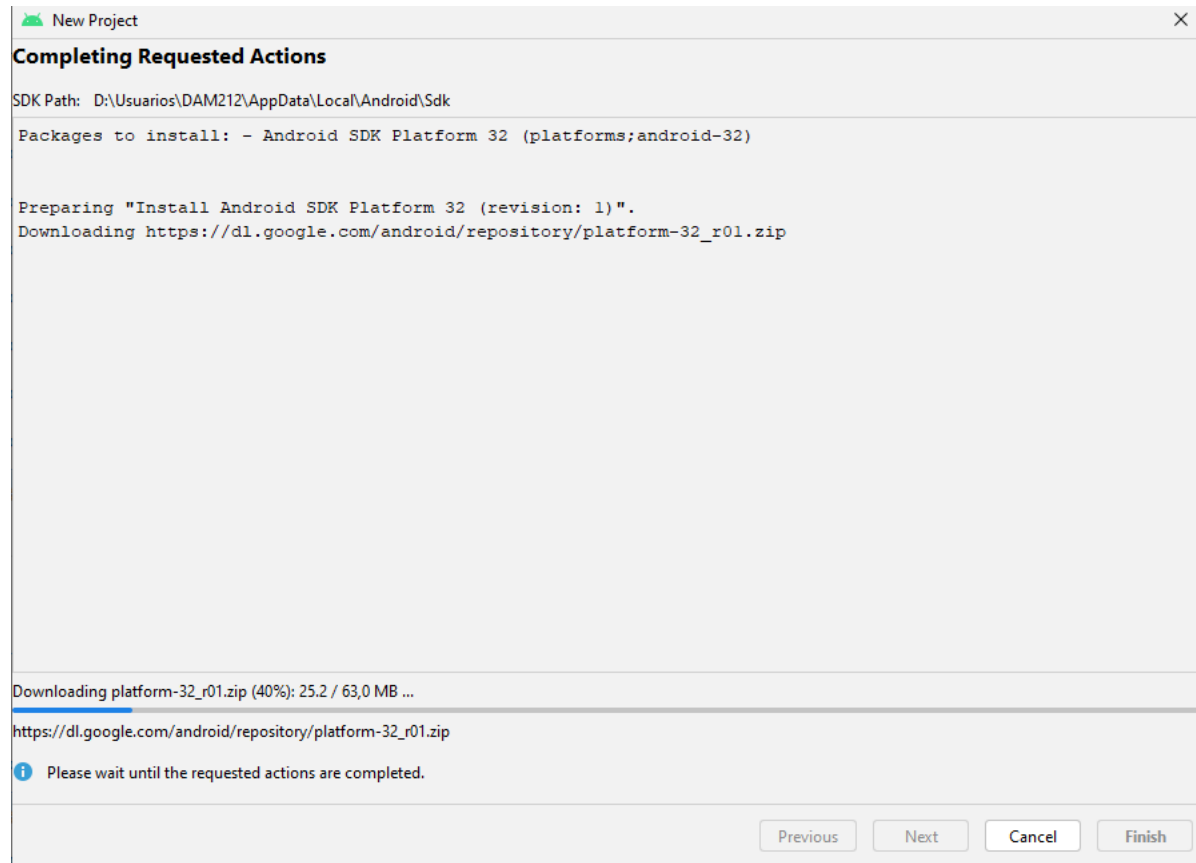
Seleccionamos el directorio, el lenguaje Kotlin y damos un nombre a la aplicación. Vemos que la SDK escogida durante la instalación es la API 21: Lollipop..





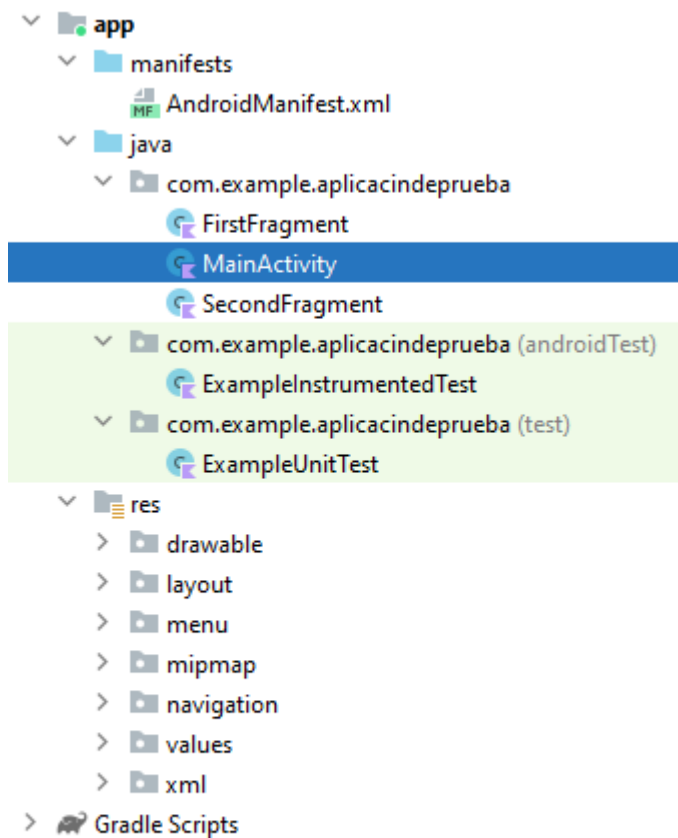
En esta pantalla nos indica también la versión mínima de SDK que admitirá la aplicación, que quedará definida en build.gradle. Vemos que la aplicación correrá en dispositivos a partir de la versión 5 de Android. En este momento el programa comienza a descargar e instalar todos los paquetes requeridos para este tipo de proyecto.





Cuando está todo listo, se abre el proyecto en blanco y nos encontramos con esta estructura de programa:





Carpetas:

Manifiesto: en esta carpeta está el archivo AndroidManifest.xml : un archivo esencial que describe las características de la aplicación para las herramientas de creación de Android, el sistema operativo Android y Google Play.



Java: en esta carpeta están los archivos de código, como hemos seleccionado basic activity nos encontramos con una clase main y dos fragments, que son porciones reutilizables de código.

Recursos(res): en esta carpeta van todos los recursos sin código. Aquí guardaremos todo el material multimedia que vayamos a añadir a la aplicación

- Imagen

- Layouts

- etcétera

Gradle Scripts: Gradle es un paquete de herramientas para automatizar y administrar el proceso de compilación y definir configuraciones de compilación personalizadas y flexibles.

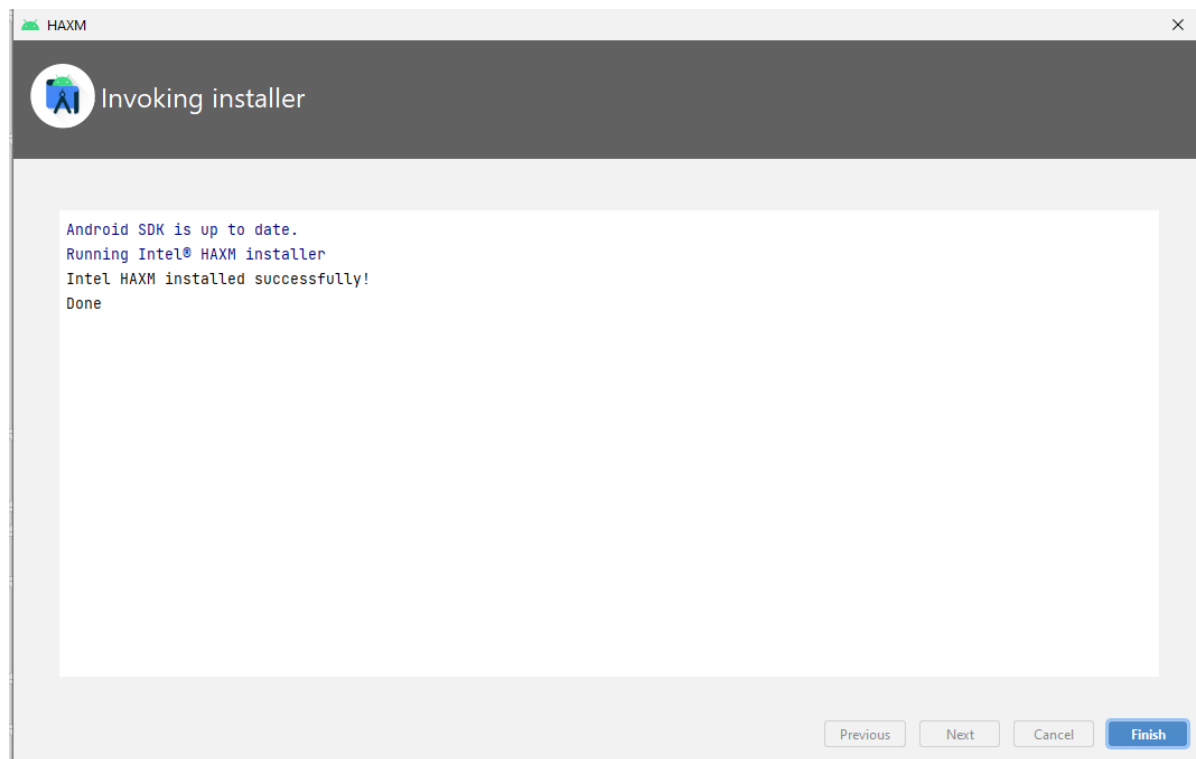
Estos no son todos los archivos del proyecto, si activamos la vista project encontraríamos el resto de archivos ocultos como la carpeta build en la que se almacenan los resultados de emulación o la carpeta libs en la que se guardan las bibliotecas privadas.



Crear un emulador

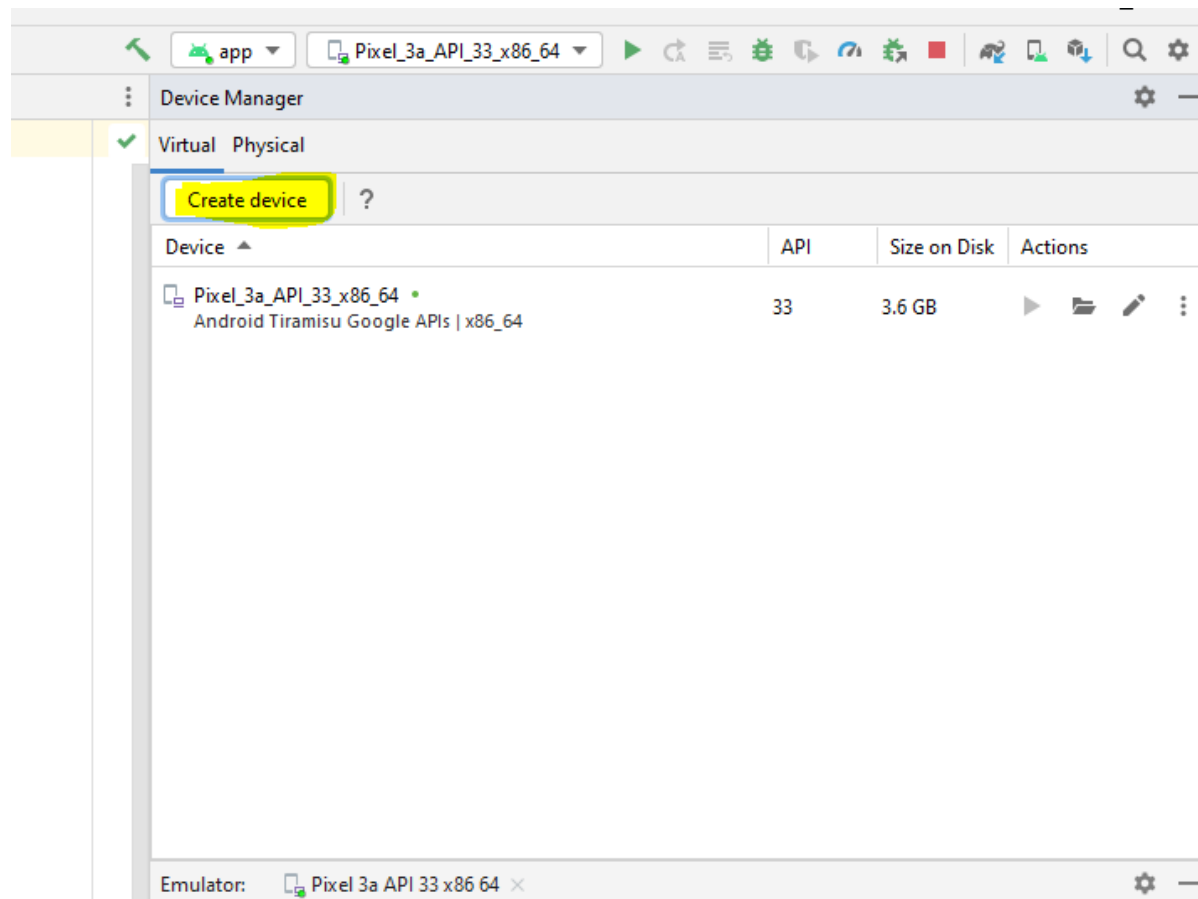
En la parte superior derecha de la pantalla principal tenemos un combo desplegable con los emuladores disponibles. Para añadir uno nuevo el programa nos pide que instalemos Haxm.





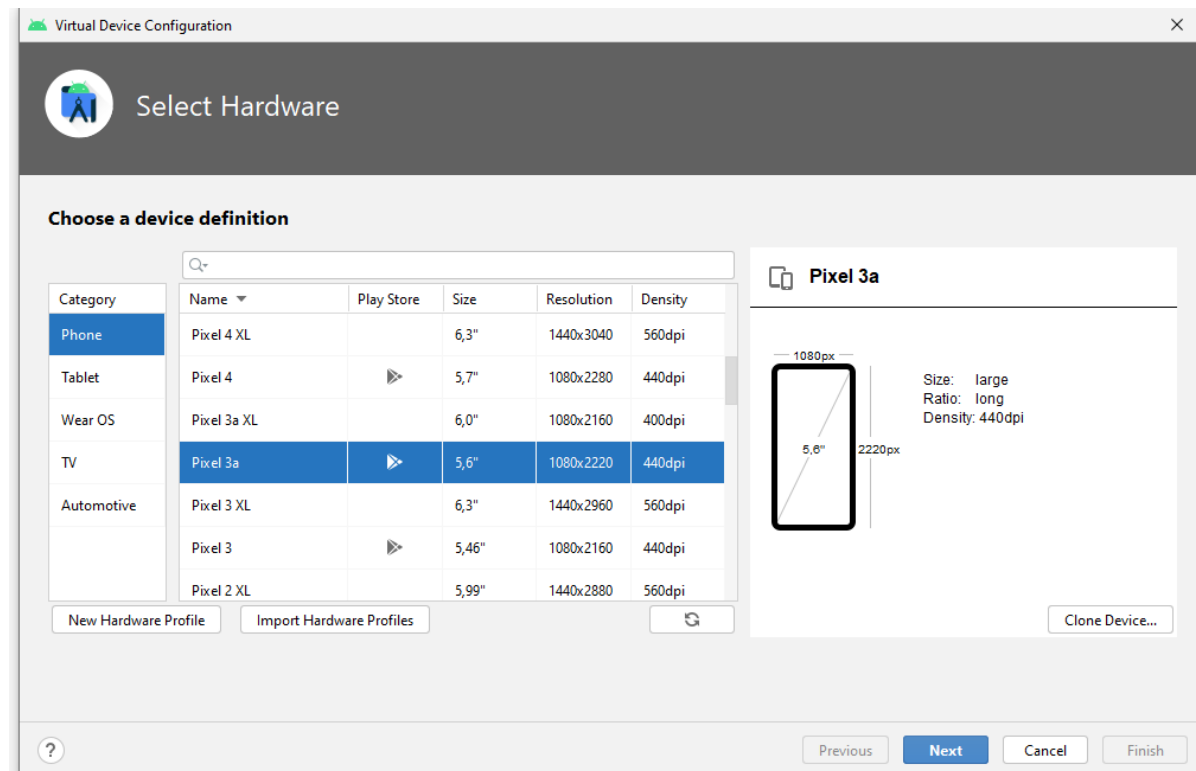
Una vez instalado pulsamos el botón Create device para empezar a construir un emulador con las características que necesitemos en función del proyecto.





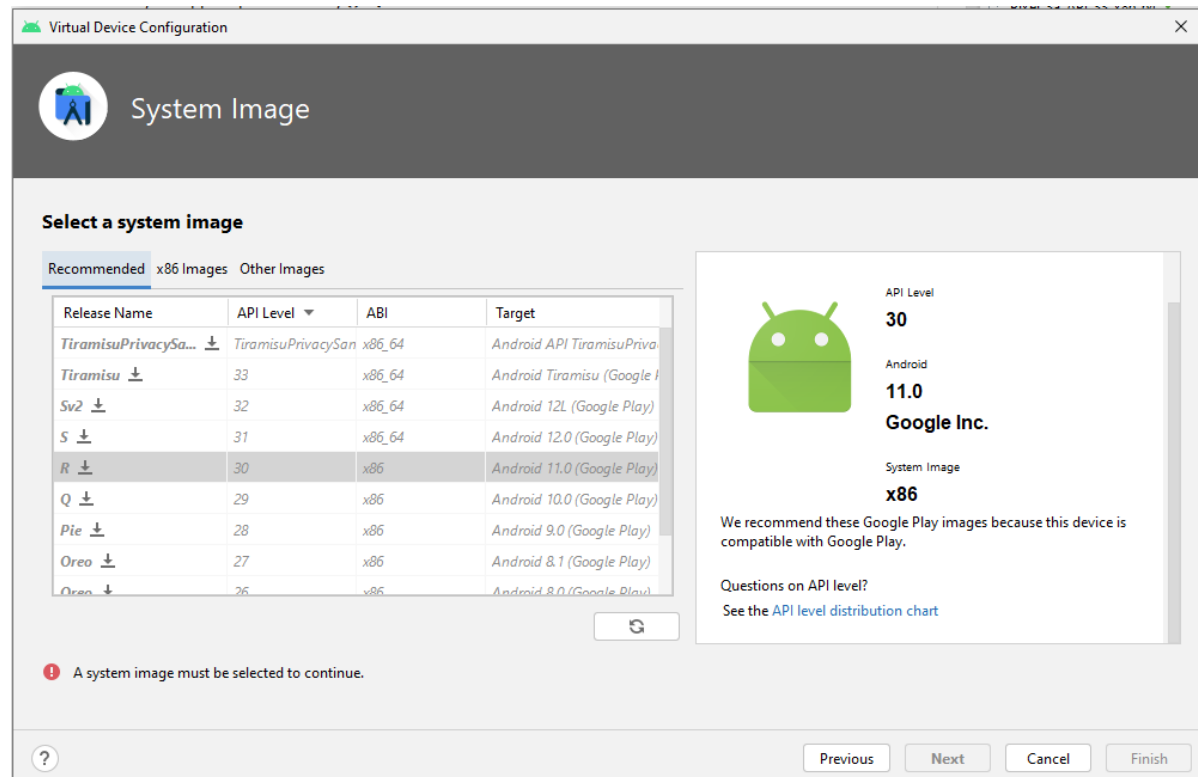
En este caso escogemos Pixel 3, que , por experiencia, sabemos que muestra bastante bien la mayoría de las aplicaciones de tamaño medio.





Escogemos, una Sdk ni demasiado antigua para estar desfasada, ni demasiado nueva para darnos problemas. En este caso escogemos la versión 30, que será compatible con el 50.3% de los dispositivos.





Descargamos la imagen de sistema.



Virtual Device Configuration

System Image

Select a system image

Recommended x86 Images Other Images

Release Name	API Level	ABI	Target
TiramisuPrivacySan...	TiramisuPrivacySan	x86_64	Android API TiramisuPrivac...
Tiramisu	33	x86_64	Android Tiramisu (Google Pl...
Sv2	32	x86_64	Android 12L (Google Play)
S	31	x86_64	Android 12.0 (Google Play)
R	30	x86	Android 11.0 (Google Play)
Q	29	x86	Android 10.0 (Google Play)
Pie	28	x86	Android 9.0 (Google Play)
Oreo	27	x86	Android 8.1 (Google Play)
Oreo	26	x86	Android 8.0 (Google Play)

A system image must be selected to continue.

SDK Quickfix Installation

SDK Component Insta

Completing Requested Actions

SDK Path: D:\Usuarios\DAM212\AppData\Local\Android

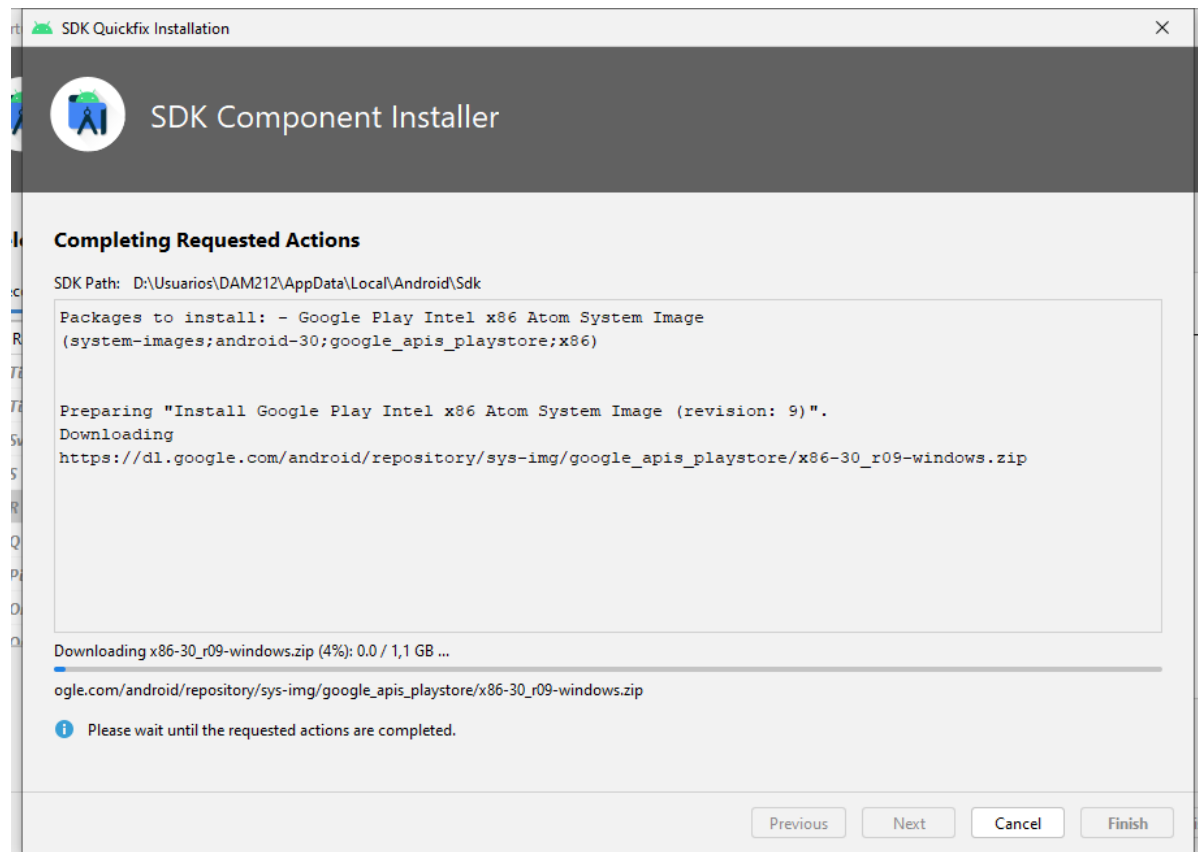
Packages to install: - Google Play Int
(system-images;android-30;google_apis_

Preparing "Install Google Play Intel >
Downloading
<https://dl.google.com/android/repository>

Downloading x86-30_r09-windows.zip (24%): 0.3 / 1,1 GB .
[ogle.com/android/repository/sys-img/google_apis_playst](https://dl.google.com/android/repository/sys-img/google_apis_playst)

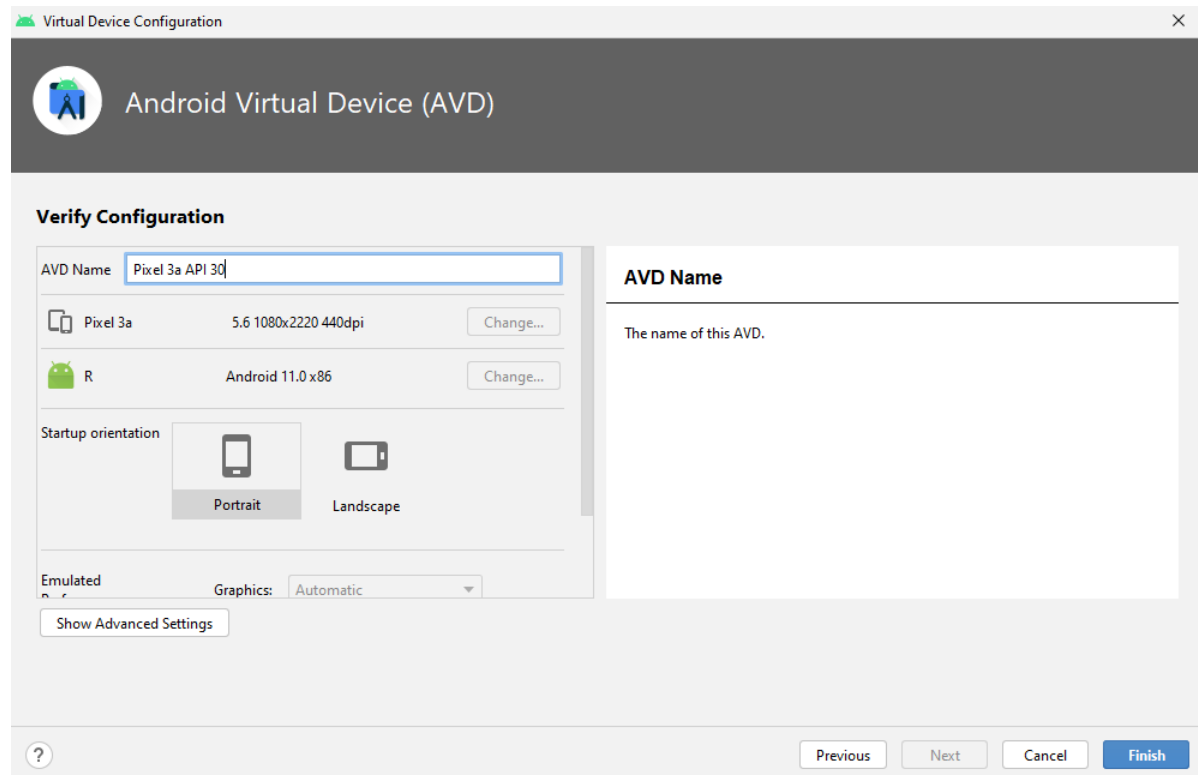
Y la descarga comienza automáticamente.





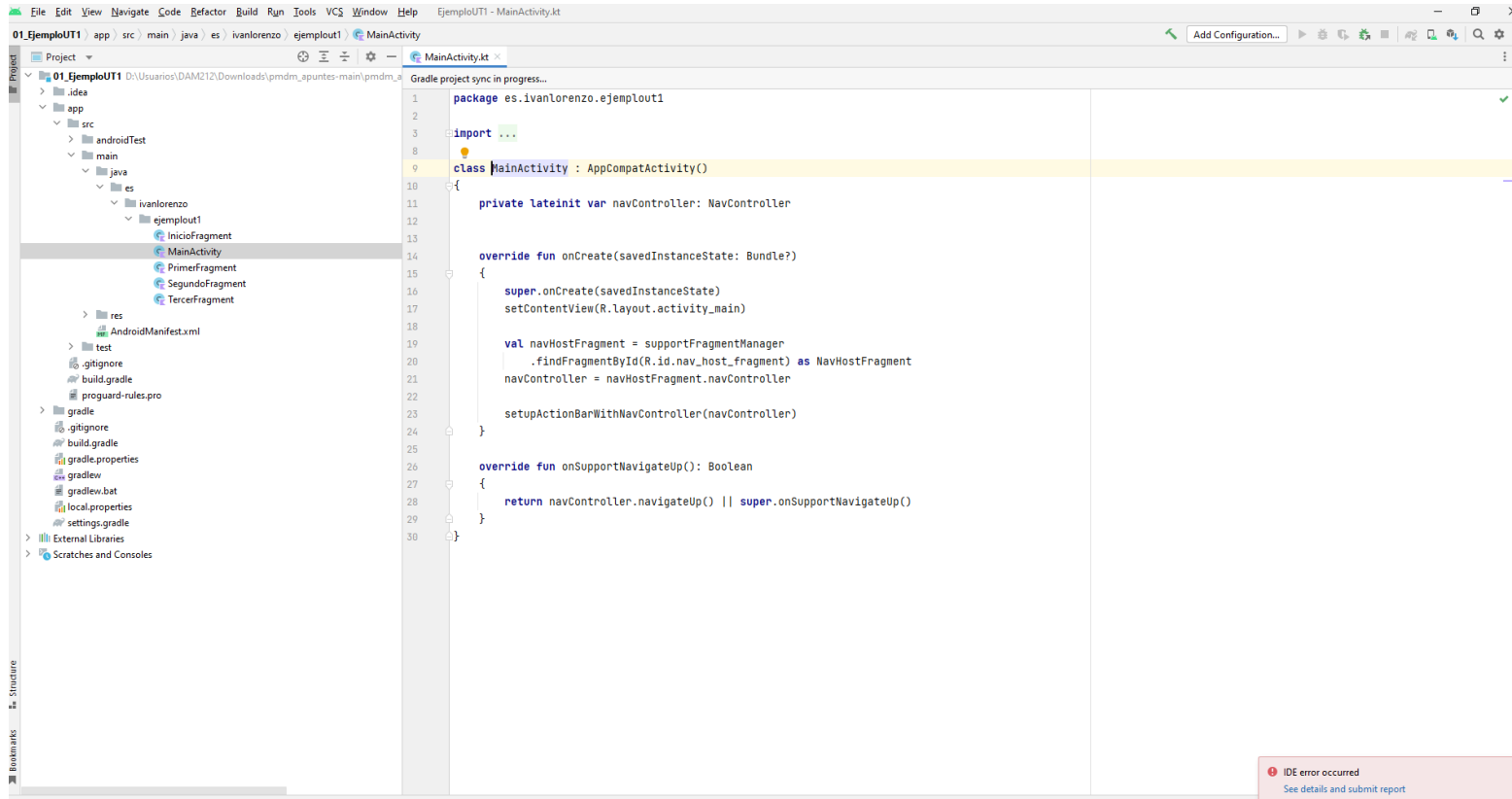
Una vez terminado este proceso podemos renombrar el dispositivo para reconocerlo y definir la orientación inicial.





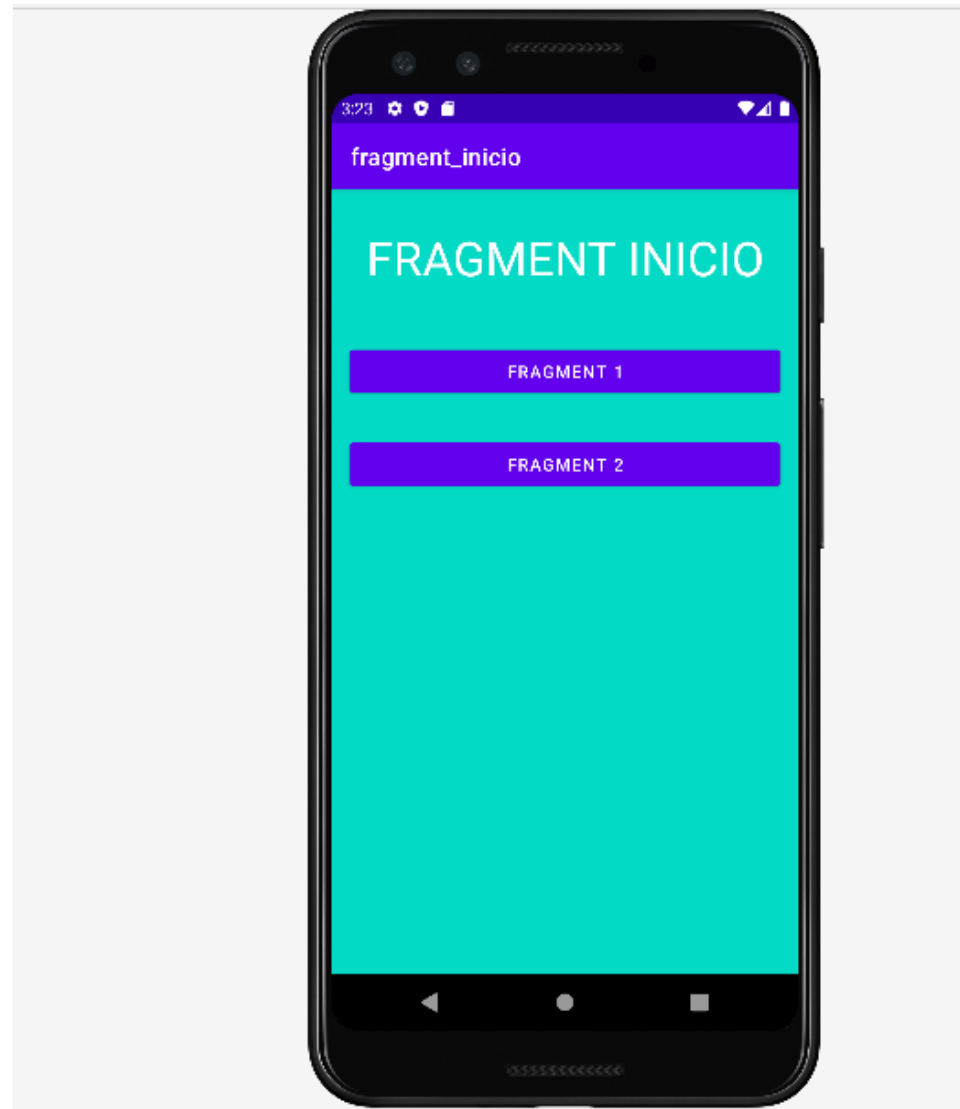
Abrir un proyecto existente y probarlo:

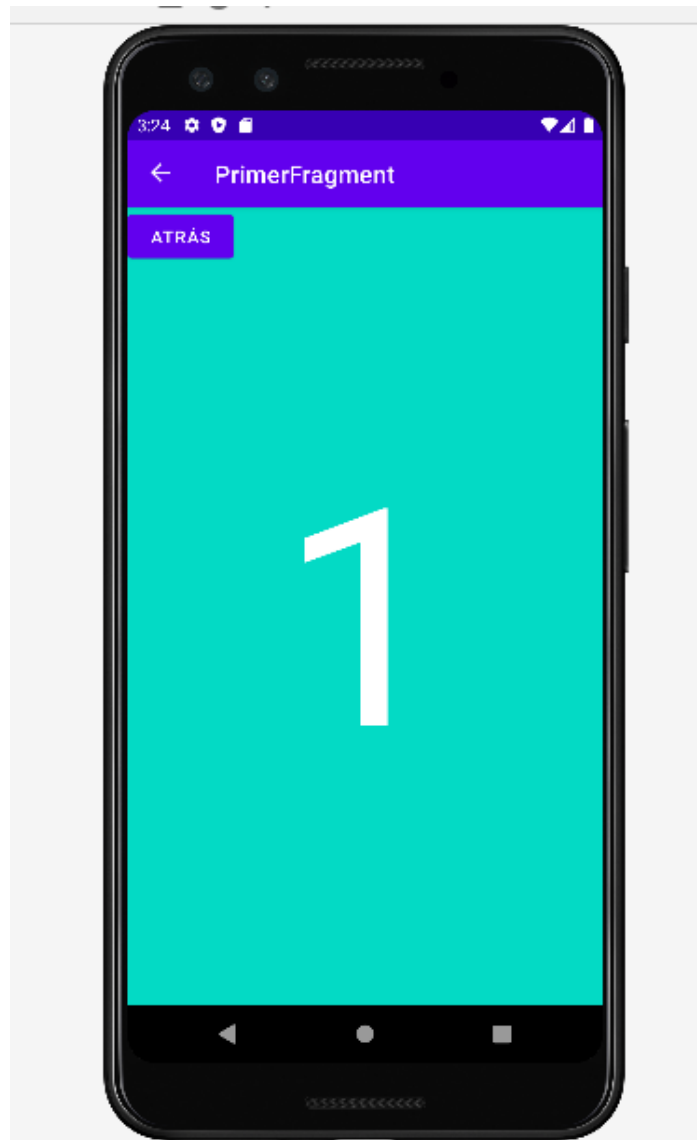
Abrimos el ejemplo y observamos que tiene una activity y varios fragments: uno de inicio y tres más.

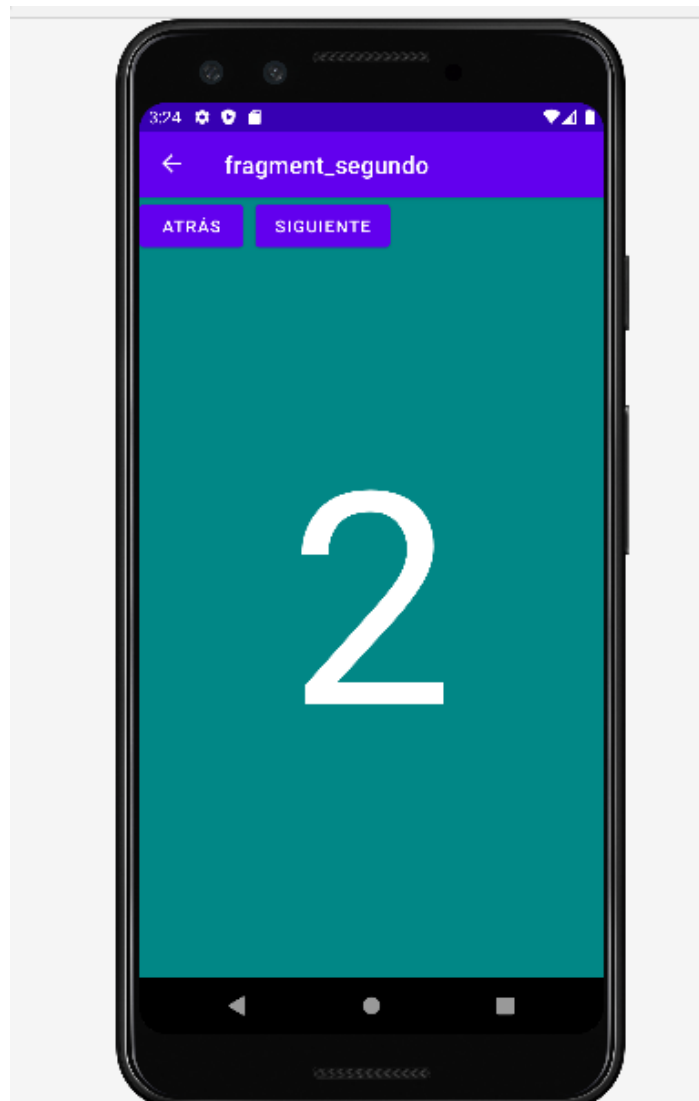


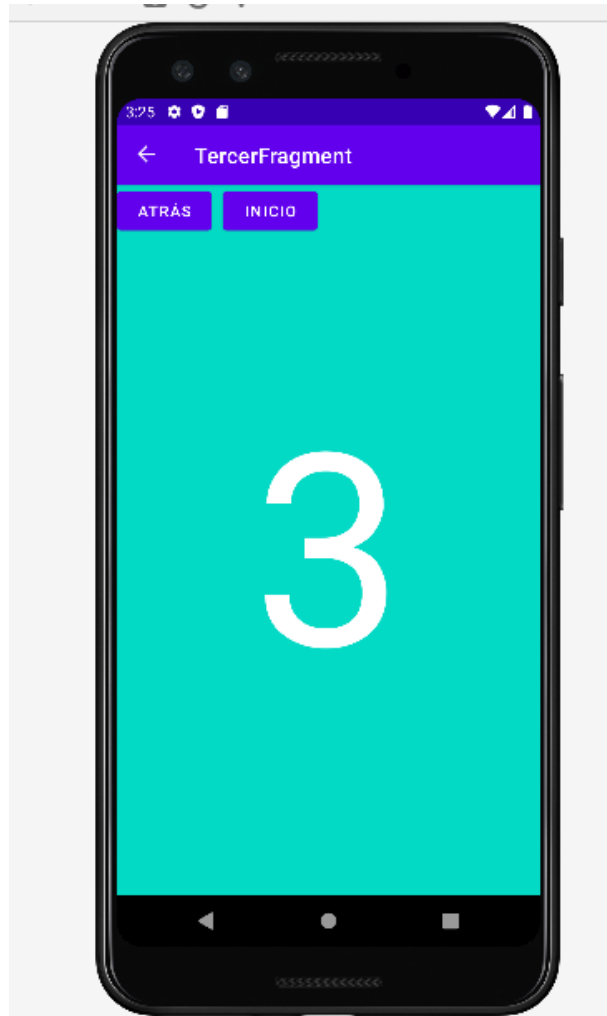
Navegamos a través de los fragments mediante botones.











Ciclo de vida de una Activity

`onCreate(Bundle)`: Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase `Bundle`), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

`onStart()`: Nos indica que la actividad está a punto de ser mostrada al usuario.

`onResume()`: Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es el momento ideal para comenzar las consultas a las bases de datos, y el `onPause()` para pausarlas. De esta forma se reduce el consumo de datos por parte de la aplicación hacia el usuario, lo que es una buena práctica bien valorada.

`onPause()`: Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad la sustituye. Es cuando se detienen las animaciones, música o el momento de almacenar los datos que estaban en edición.

`onStop()`: La actividad ya no va a ser visible para el usuario.

`onRestart()`: Indica que la actividad va a volver a ser representada después de haber pasado por `onStop()`.

`onDestroy()`: Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al método `finish()`.

Funcionamiento en una aplicación en Android

Lo veremos a través de éste código en el main: que muestra un mensaje en el Run según en qué estado se encuentre la aplicación.





```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        Log.i( tag: "Ciclo de Vida", msg: "método onCreate")  
    }  
    override fun onStart() {  
        super.onStart()  
        Log.i( tag: "Ciclo de Vida", msg: "metodo onStart")  
    }  
    override fun onResume() {  
        super.onResume()  
        Log.i( tag: "Ciclo de Vida", msg: "metodo onResume")  
    }  
    override fun onPause() {  
        super.onPause()  
        Log.i( tag: "Ciclo de Vida", msg: "metodo onPause")  
    }  
    override fun onStop() {  
        super.onStop()  
        Log.i( tag: "Ciclo de Vida", msg: "metodo onStop")  
    }  
    override fun onRestart() {  
        super.onRestart()  
        Log.i( tag: "Ciclo de Vida", msg: "metodo onRestart")  
    }  
}
```




```

override fun onRestart() {
    super.onRestart()
    Log.i( tag: "Ciclo de Vida", msg: "metodo onRestart")
}

override fun onDestroy() {
    super.onDestroy()
    Log.i( tag: "Ciclo de Vida", msg: "metodo onDestroy")
}

```

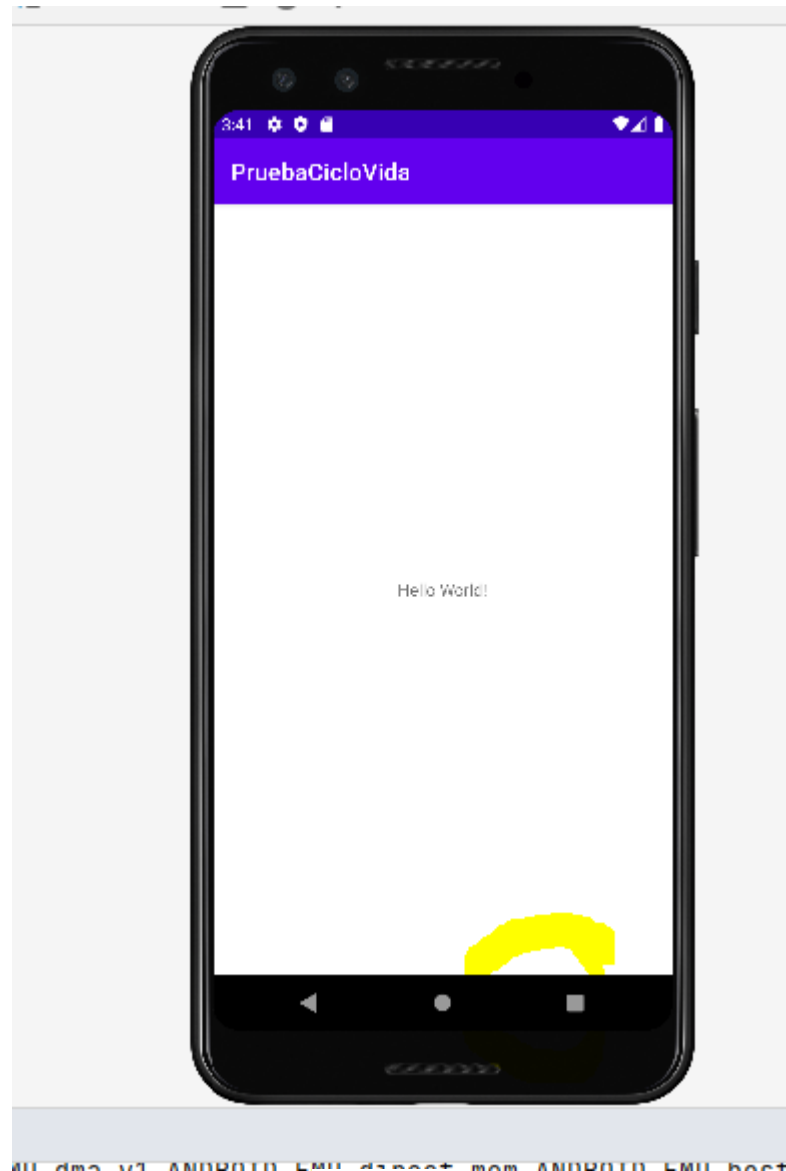
Cuando arrancamos la aplicación, se suceden el método onCreate, On Start y onResume, como vemos en la pestaña run

```

W/pruebaciclovid: Redefining intrinsic method b
D/NetworkSecurityConfig: No Network Security Co
D/NetworkSecurityConfig: No Network Security Co
D/libEGL: loaded /vendor/lib/egl/libEGL_emulati
D/libEGL: loaded /vendor/lib/egl/libGLESv1_CM_e
D/libEGL: loaded /vendor/lib/egl/libGLESv2_emul
W/pruebaciclovid: Accessing hidden method Landr
W/pruebaciclovid: Accessing hidden method Landr
I/Ciclo de Vida: método onCreate
I/Ciclo de Vida: metodo onStart
I/Ciclo de Vida: metodo onResume

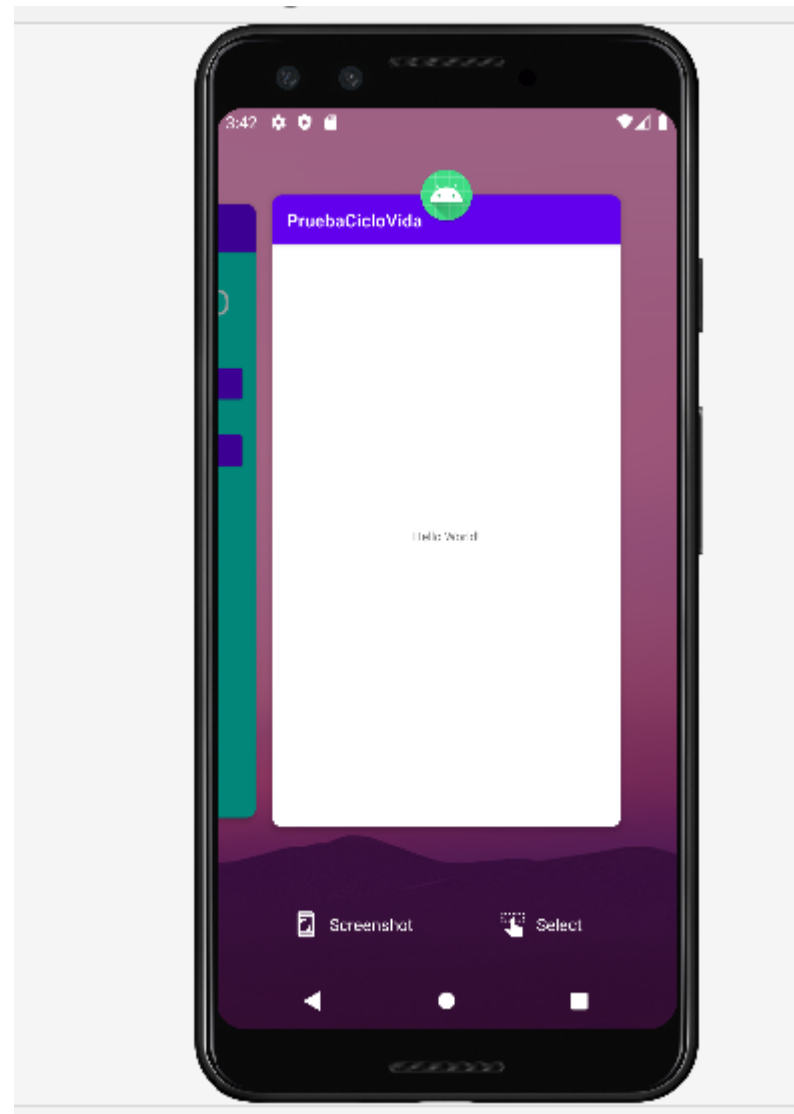
```



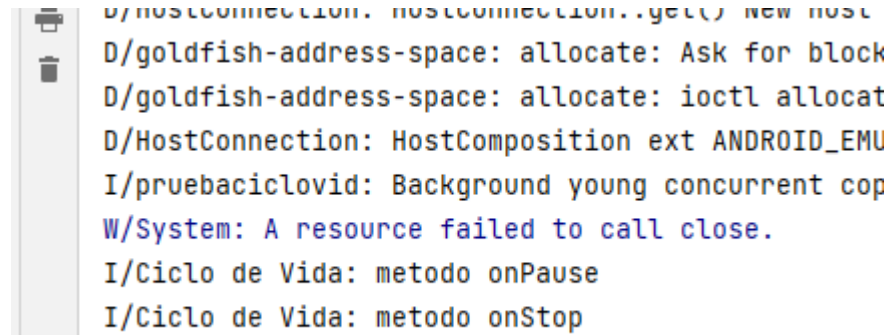


Si pulsamos en este botón veremos que la app pasa a segundo plano



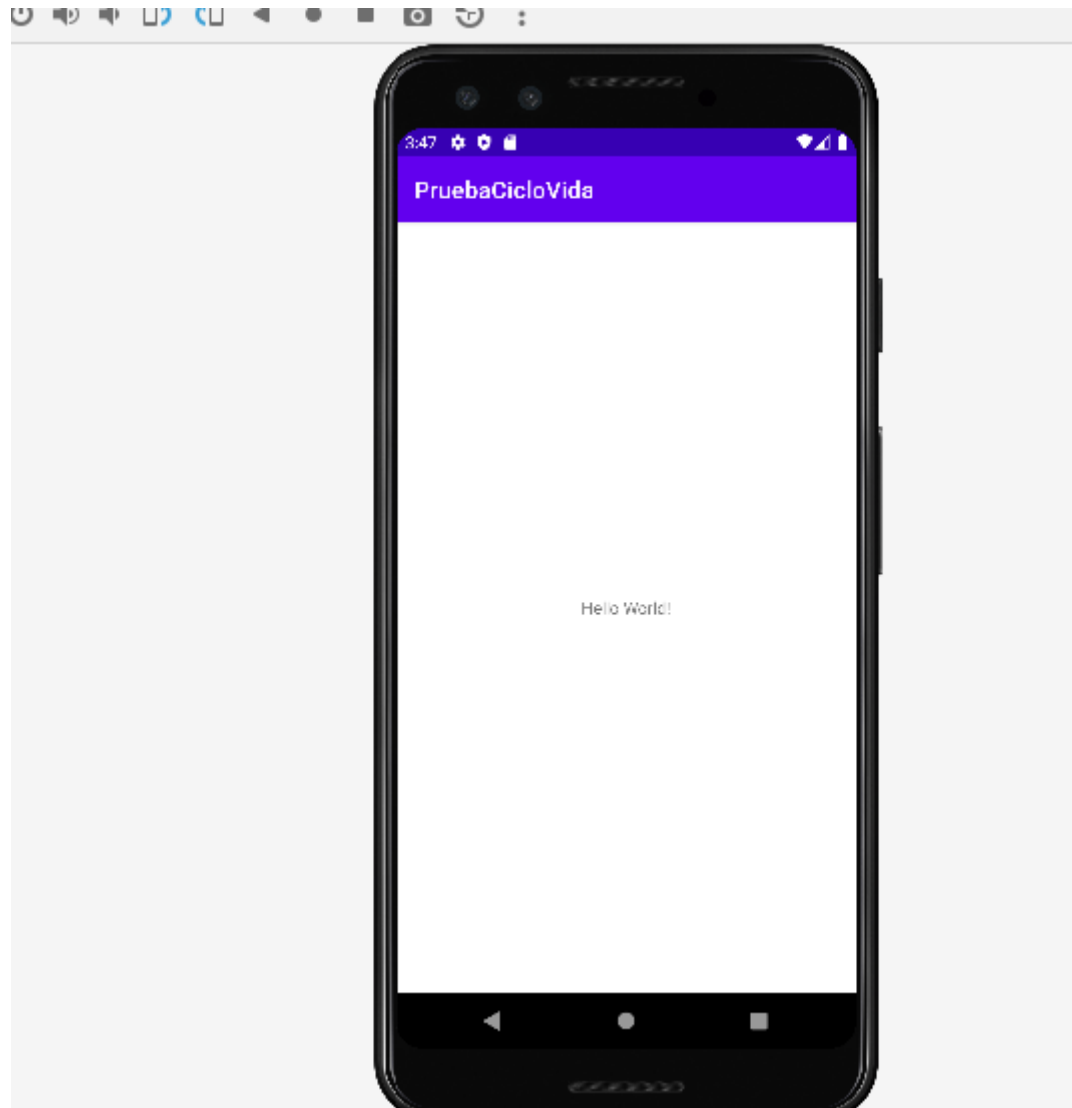


En el Run se observa que se han ejecutado los métodos on Pause y onStop

A screenshot of the Android Studio logcat window. The log shows several system messages. The most relevant ones for the context are 'I/Ciclo de Vida: metodo onPause' and 'I/Ciclo de Vida: metodo onStop', which confirm that the application's lifecycle methods were called. Other logs include messages from 'D/goldfish-address-space' about block allocation and 'W/System' about a resource failure to call close.

```
D/HostConnection: HostConnection.get() new HostConnection()
D/goldfish-address-space: allocate: Ask for block from kernel pool
D/goldfish-address-space: allocate: ioctl allocate succeeded
D/HostConnection: HostComposition ext ANDROID_EMU not registered in AID_CONFIG
I/pruebaciclovida: Background young concurrent copy
W/System: A resource failed to call close.
I/Ciclo de Vida: metodo onPause
I/Ciclo de Vida: metodo onStop
```





Si haciendo click regresamos a la aplicación vemos como el main ejecuta



```
I/pruebaciclovid: Background young concur  
W/System: A resource failed to call close  
I/Ciclo de Vida: metodo onPause  
I/Ciclo de Vida: metodo onStop  
I/Ciclo de Vida: metodo onRestart  
I/Ciclo de Vida: metodo onStart  
I/Ciclo de Vida: metodo onResume
```





```
override fun onRestart() {
```






```
w/System: A resource failed to call close
I/Ciclo de Vida: metodo onPause
I/Ciclo de Vida: metodo onStop
I/Ciclo de Vida: metodo onRestart
I/Ciclo de Vida: metodo onStart
I/Ciclo de Vida: metodo onResume
I/Ciclo de Vida: metodo onPause
I/Ciclo de Vida: metodo onStop
```








```
W/System: A resource failed to call close
I/Ciclo de Vida: metodo onPause
I/Ciclo de Vida: metodo onStop
I/Ciclo de Vida: metodo onRestart
I/Ciclo de Vida: metodo onStart
I/Ciclo de Vida: metodo onResume
I/Ciclo de Vida: metodo onPause
I/Ciclo de Vida: metodo onStop
I/Ciclo de Vida: metodo onRestart
I/Ciclo de Vida: metodo onStart
I/Ciclo de Vida: metodo onResume
```





Si pulsamos el botón de salir (triángulo), la aplicación ejecutará onDestroy()



```
D/goldfish-address-space: allocate: Ask for block of
D/goldfish-address-space: allocate: ioctl allocate re
D/HostConnection: HostComposition ext ANDROID_EMU_CHE
W/System: A resource failed to call close.
I/Ciclo de Vida: metodo onPause
I/Ciclo de Vida: metodo onStop
I/Ciclo de Vida: metodo onDestroy
```

