

Práctica 1.

Instalando Android Studio y creando la primera aplicación.

1. Resultados de aprendizaje y criterios de evaluación.

CE1c. Se han instalado, configurado y utilizado entornos de trabajo para el desarrollo de aplicaciones para dispositivos móviles.

CE1h. Se han utilizado emuladores para comprobar el funcionamiento de las aplicaciones.

2. Instalando Android Studio.

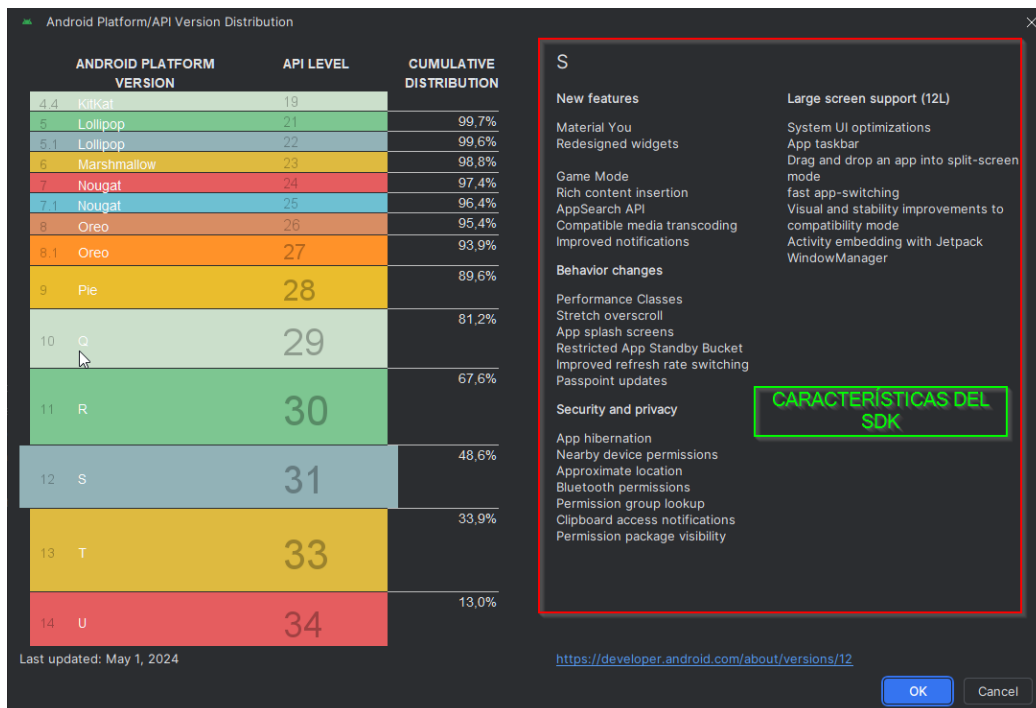
La instalación de Android Studio es una tarea sencilla, simplemente ir a la url: <https://developer.android.com/studio>, descargar, descomprimir e iniciar (en el caso de Linux Mint), en el caso de Windows es necesario realizar el proceso de instalación.

3. Configurando los SDK.

Es necesario descargar el SDK concreto que se desee utilizar, por ejemplo, si se quiere que la aplicación sea compatible solo con la última versión (14) o dar retrocompatibilidad para abarcar más mercado, como puede ser desde la versión 10.

En la instalación por defecto se incluye el SDK 34

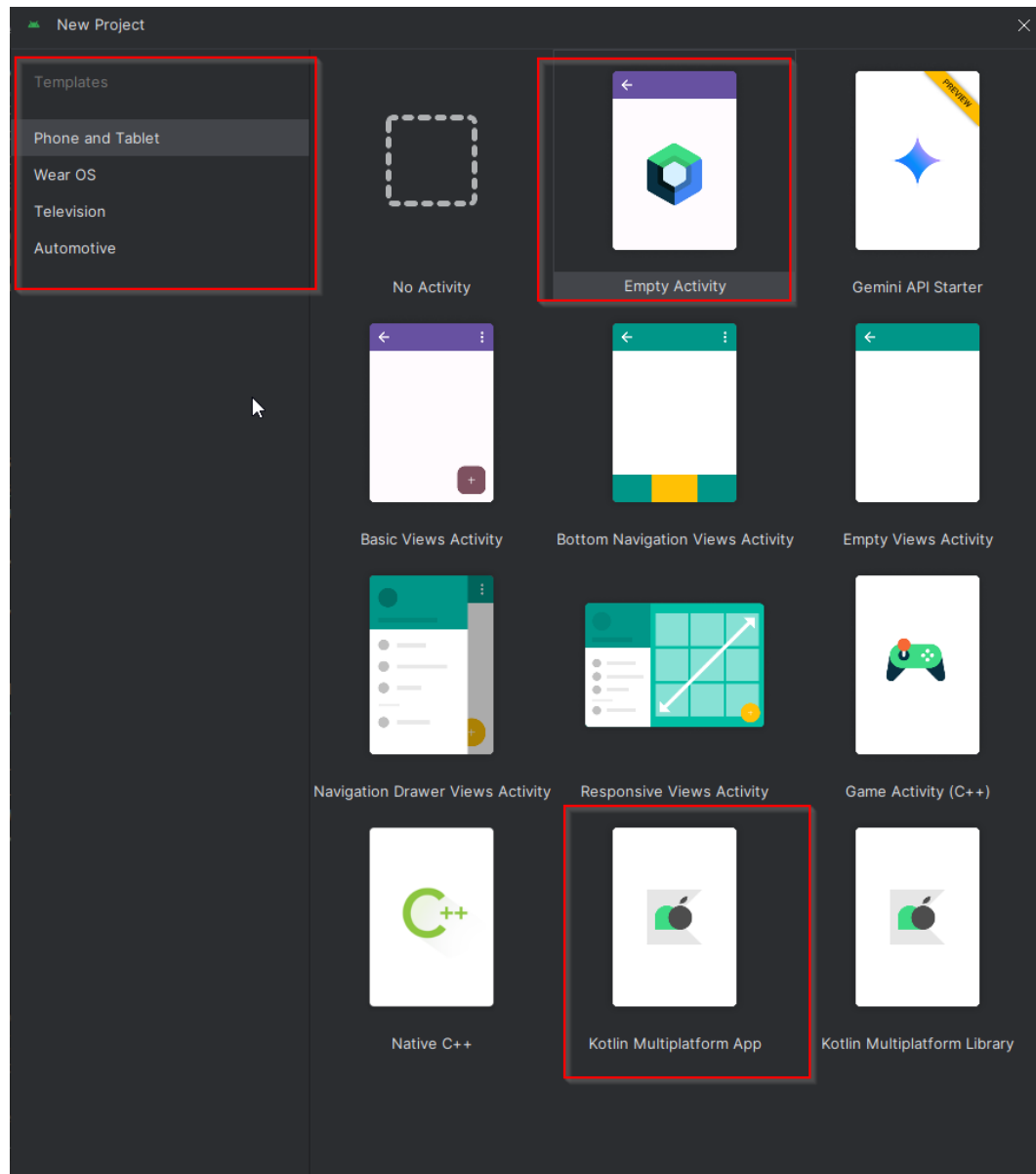
En el siguiente gráfico que facilita Android Studio al crear un nuevo proyecto, con el porcentaje de uso y las características de la versión.



4. Creando el primer proyecto.

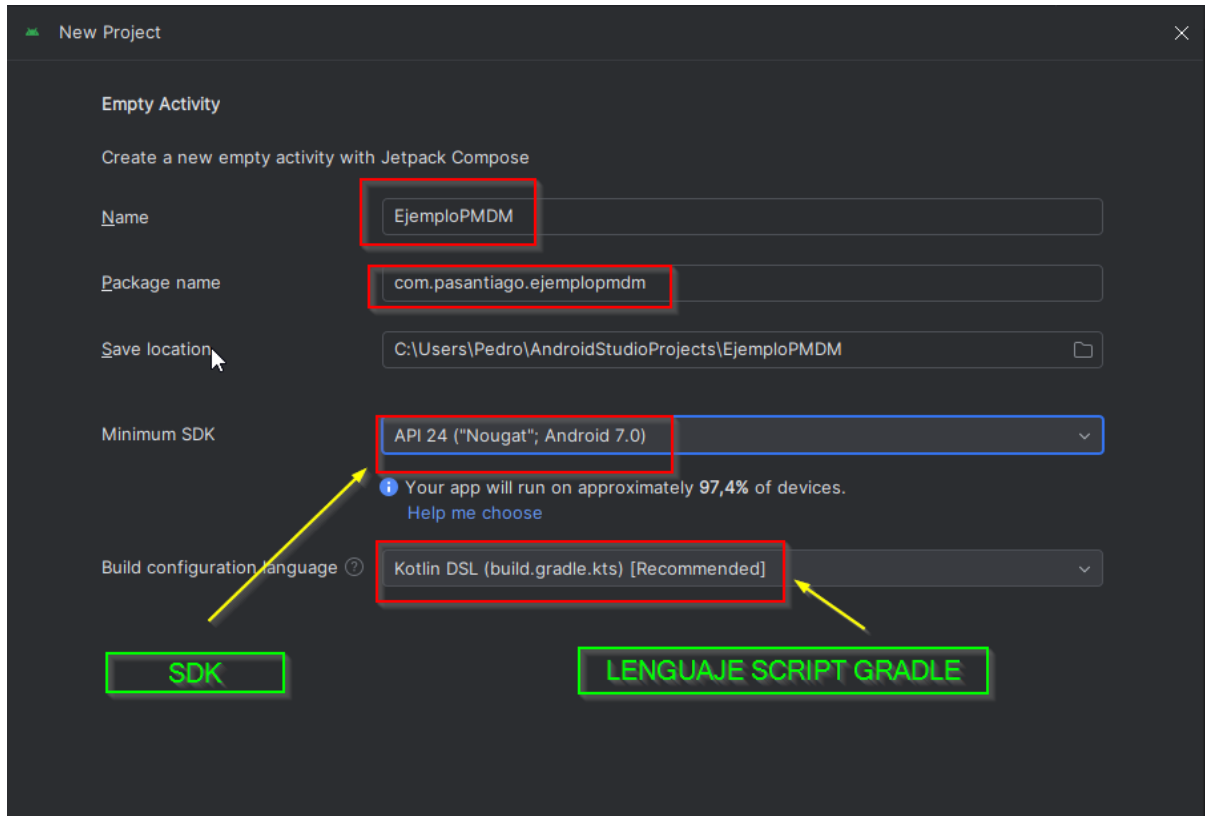
La creación de un proyecto es sencilla usando el asistente, internamente utiliza el software de gestión y automatización de código “Gradle”. Ir al menú “File” -> “New Project”, apareciendo diferentes “templates” de proyectos para teléfonos y tabletas, relojes, televisores e industria del automóvil.

En cada de las secciones se dispone de “esqueletos” para los tipos de aplicaciones más comunes, por ejemplo, en el caso de móviles:



En este caso se utiliza la plantilla “Empty Activity”, apareciendo un nuevo paso en que se ha de configurar el proyecto indicando:

- Nombre de la aplicación.
- Paquete que se generará.
- Localización.
- SDK mínimo necesario en el dispositivo cliente para funcionar.
- Lenguaje de los “script” de construcción usado por “Gradle”, por ejemplo, subida de código a GitHub o realización de pruebas automáticas en varios emuladores.

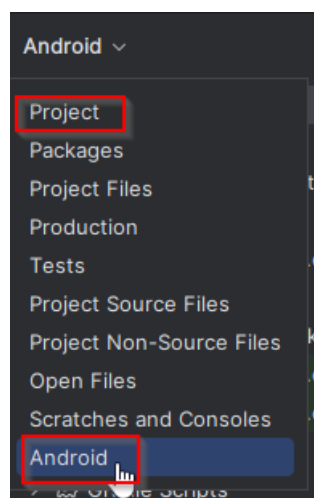


En el caso de no estar instalado el SDK para la versión seleccionada.

5. Analizando la estructura del proyecto.

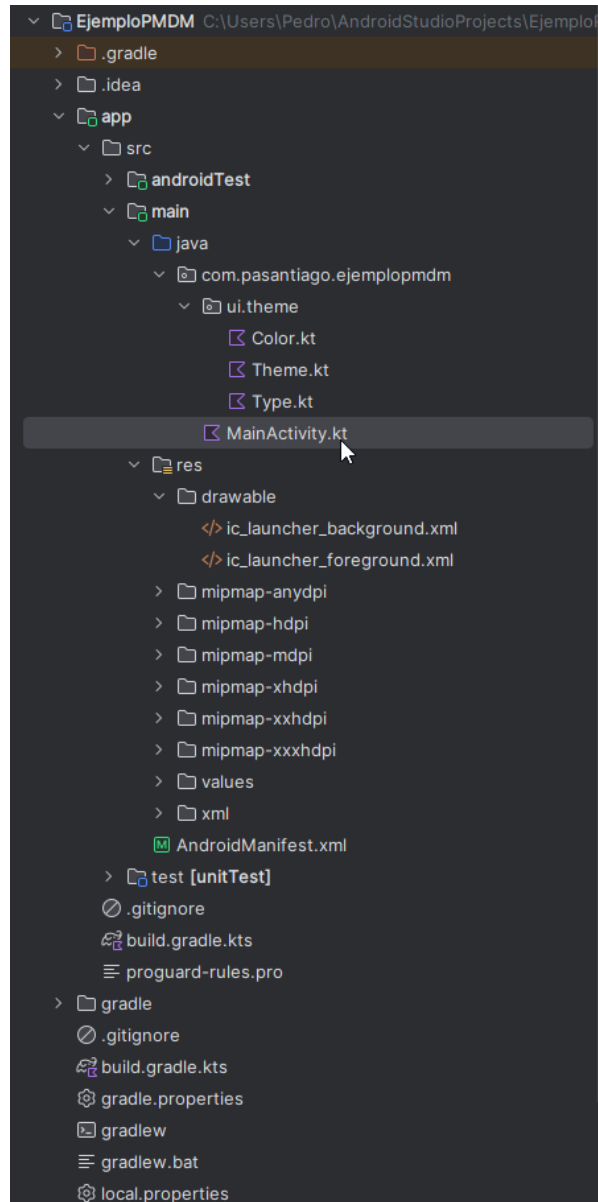
Ya se ha creado el proyecto, y en la parte izquierda del IDE aparece la estructura de este (tarda un poco la primera vez al descargar todo lo necesario para descargar todas las herramientas de "Gradle" en analizar los ficheros creados y mostrarlo).

Se ha de diferenciar entre la estructura del proyecto que muestra el IDE (Android) y la estructura real en el sistema de archivos (Project).



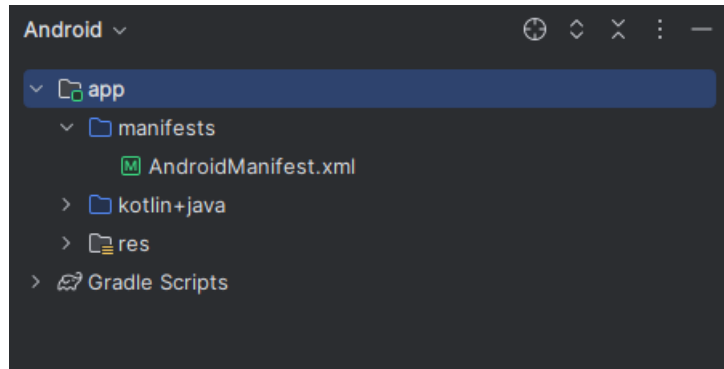
En "project" se puede observar además de los ficheros tal y como se encuentran en el sistema de archivo, se tiene (extraído a partir de la documentación oficial):

- .gradle: Directorio oculto con ficheros necesarios para construir la aplicación.
- .idea: Directorio con ficheros creado por el IDE para gestión interna.
- App: Directorio con el nombre de la aplicación o módulo, en el que se encuentran el código fuente, los recursos, las pruebas, las librerías y el proyecto compilado entre otros.
 - Build/: Contiene resultados de compilación.
 - Libs/ : Contiene bibliotecas privadas.
 - Src/: Contiene todos los archivos de código y recursos para el módulo en los siguientes subdirectorios:
 - AndroidTest/: Contiene código para las pruebas de instrumentación que se ejecutan en un dispositivo Android. Para obtener más información, consulta Cómo realizar pruebas en Android Studio.
 - Main/: Contiene los archivos de conjunto de orígenes "principales": el código y los recursos de Android compartidos por todas las variantes de compilación (los archivos para otras variantes de compilación residen en directorios del mismo nivel, como src/debug/ para el tipo de compilación de depuración).
 - AndroidManifest.xml: Describe la naturaleza de la aplicación y cada uno de sus componentes.
 - Java/: Contiene fuentes de código Kotlin o Java, o ambos, si tu app tiene código fuente Kotlin y Java.
 - Kotlin/: Contiene solo fuentes de código Kotlin.
 - Res/: Contiene recursos de aplicación, como archivos de elementos de diseño y de cadenas de IU. Para obtener más información, consulta la descripción general de los recursos de las apps.
 - Assets/: Contiene archivos para compilar en un archivo APK tal como está. Por ejemplo, es una buena ubicación para texturas y datos de juegos. Puedes explorar este directorio como un sistema de archivos típico mediante URI y leer archivos como transmisiones de bytes con AssetManager.
 - Test/: Contiene código para pruebas locales que se ejecutan en tu JVM host.
 - build.gradle o build.gradle.kts :Este archivo define las configuraciones de compilación específicas para el módulo o aplicación. build.gradle es el nombre de archivo correcto si usas Groovy como lenguaje de secuencia de comandos de compilación y es build.gradle.kts si usas la secuencia de comandos de Kotlin.
- Build: Resultado de la construcción del proyecto, teniendo en cuenta que pueden existir varios módulos o app dentro del proyecto.
- Gradle: Ficheros de configuración de Gradle



La vista de Android es más simple que la de “project”, mostrando las carpetas y ficheros relacionadas con el desarrollo, la vista contiene:

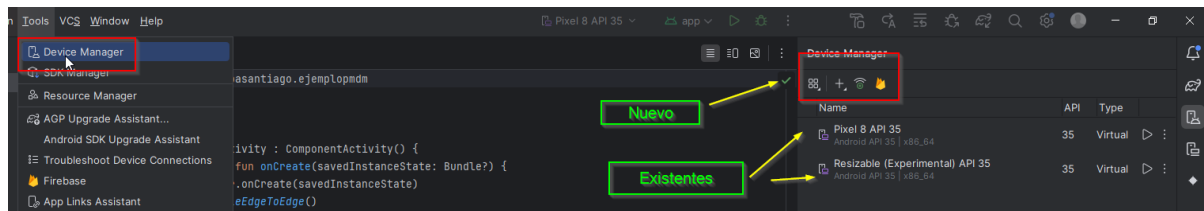
- **Manifests:** Contiene el archivo `AndroidManifest.xml`. El archivo de manifiesto describe información esencial de la aplicación para las herramientas de creación de Android, el sistema operativo Android y Google Play.
- **Java o Kotlin+java:** Contiene los archivos de código fuente de Kotlin y/o Java separados por nombres de paquetes, incluido el código de prueba JUnit.
- **Res:** Contiene todos los recursos sin código, como cadenas de IU o imágenes de mapa de bits, divididos en subdirectorios pertinentes. Para obtener más información sobre los tipos de recursos posibles, consulta Descripción general de los recursos de las apps.
- **Grade Script:** Guiones para la gestión del proyecto.



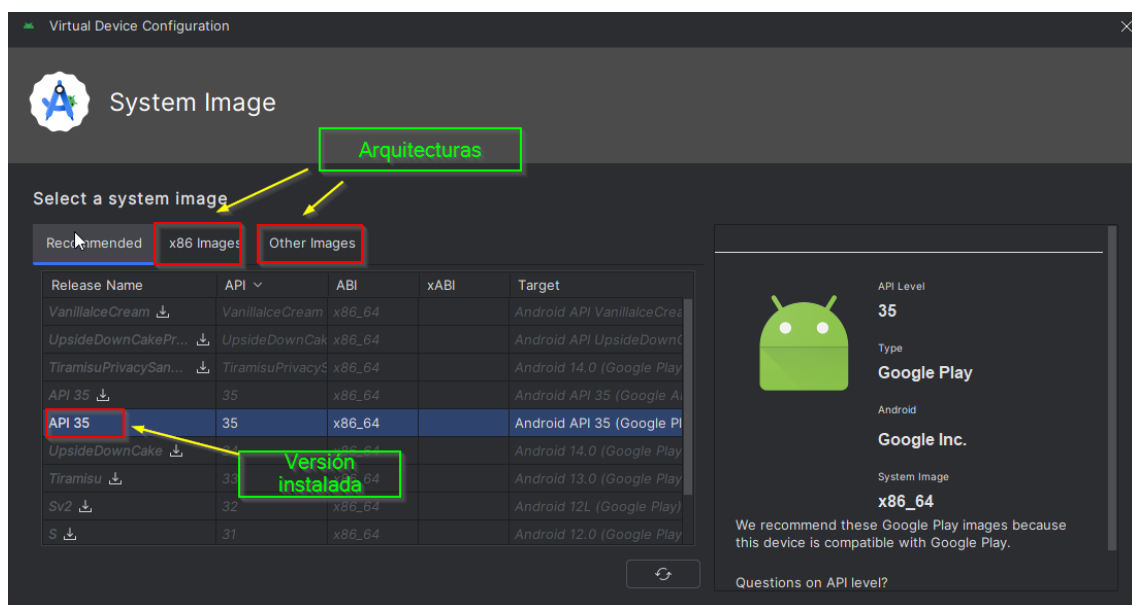
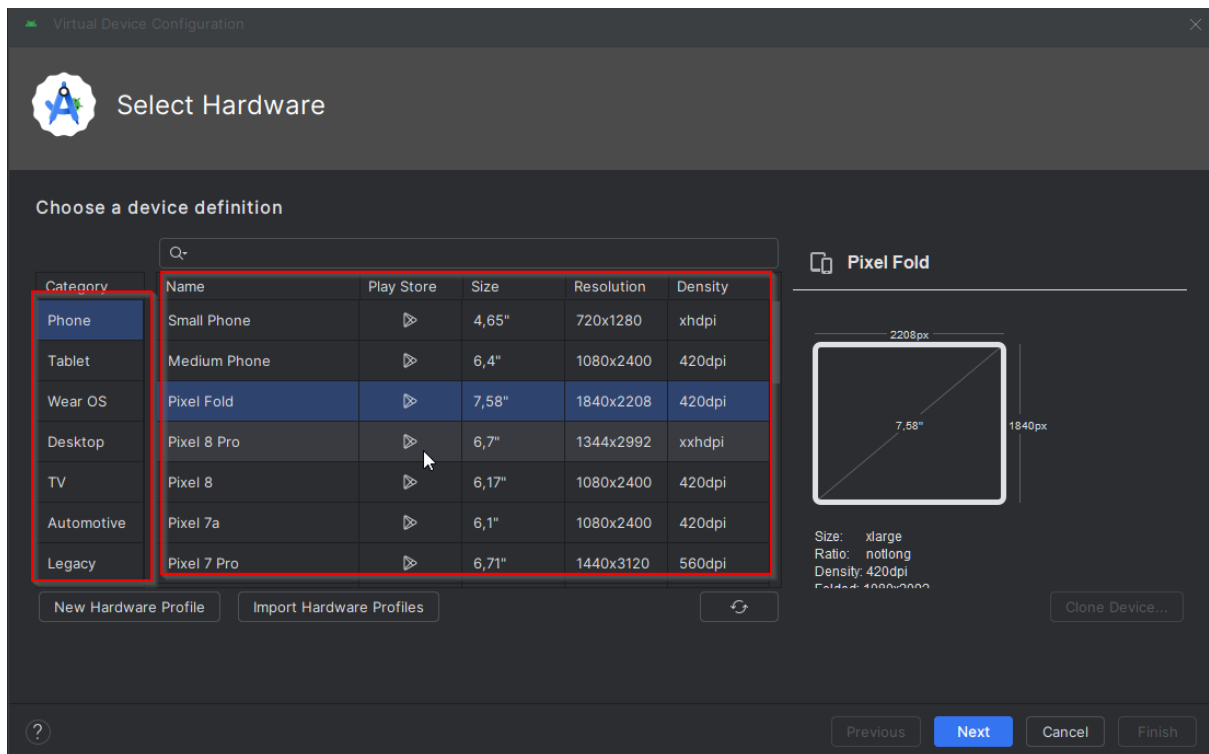
6. Configurando y ejecutando en un emulador.

Ya es posible compilar el proyecto, ahora se ha de pasar las pruebas y realizar pruebas, muy posiblemente en varios dispositivos, siendo poco probable que se disponga de una variedad de dispositivos físicos.

Para solucionarlo se puede virtualizar diferentes dispositivos en los que ejecutar el software, siendo posible gestionarlos desde varios puntos del entorno, uno de ellos es ir al menú **"Tools"** y seleccionar Device Manager:

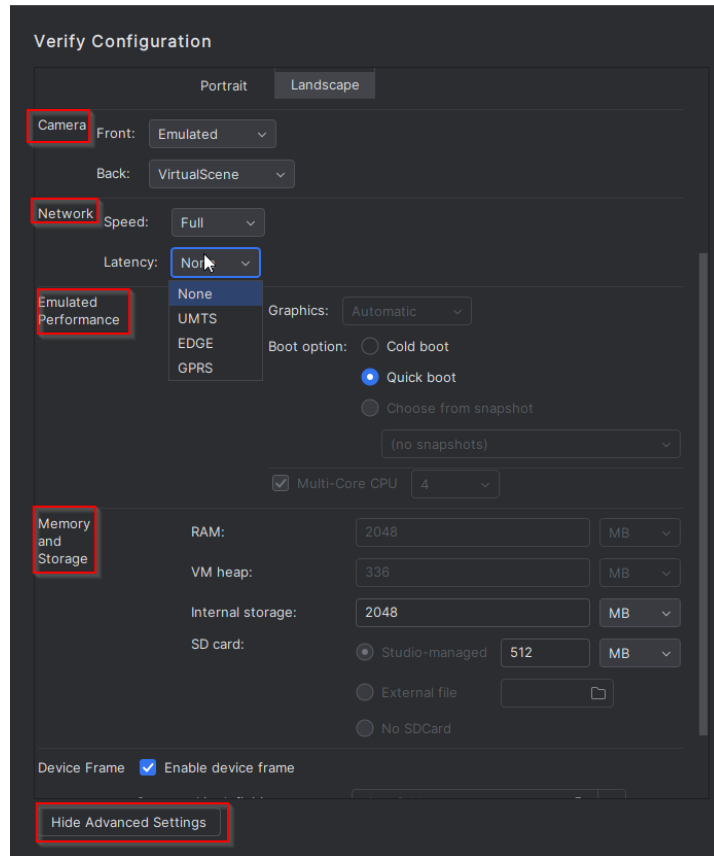


Apareciendo en pantalla una ventana en la que se pueden crear, inicial y borrar simuladores. Se selecciona un dispositivo y la versión de SDK a utilizar:

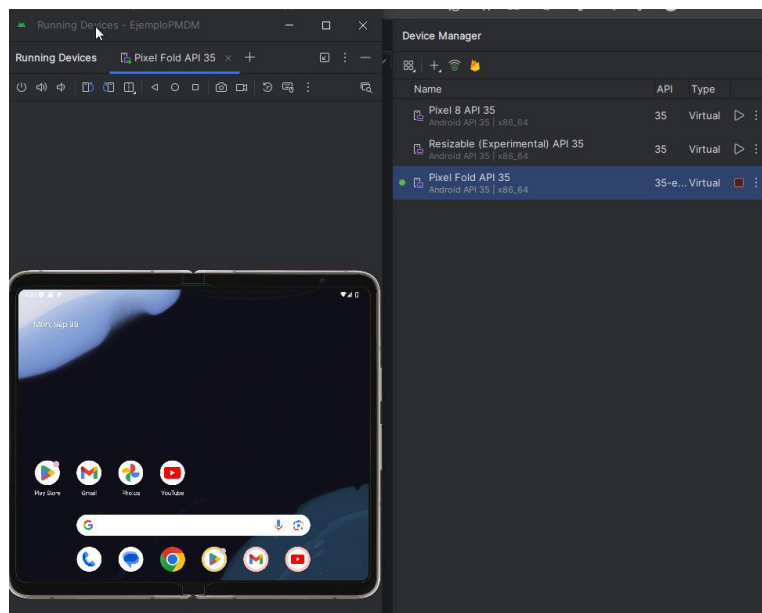


Configuración con la API 35 para x86_64, que se encuentra instalada.

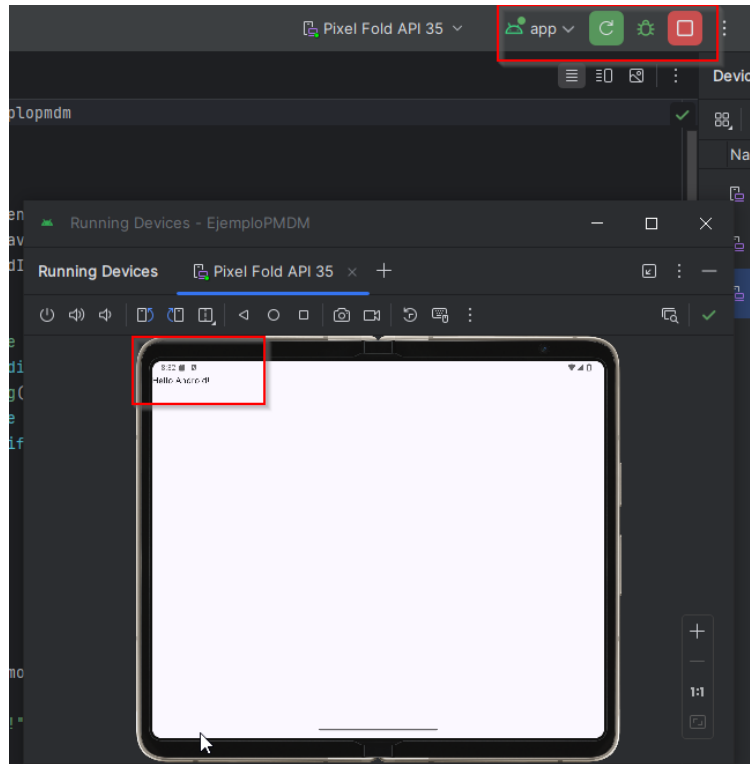
Por último, se indica el nombre, el ABI (Cada combinación de CPU y conjunto de instrucciones tiene su propia interfaz binaria de aplicación (ABI)) y otras opciones. Se puede personalizar aún más con las opciones avanzadas con las opciones de tipo de cámara, características de la red, el método de arranque, o la configuración de la memoria. También se indica el nombre del dispositivo virtual de Android (AVD), el tipo de cámara o la orientación inicial:




Ahora aparecerá en el “Administrador de Dispositivos”, pulsar “play” para iniciar el emulador (es un proceso lento):



Ya es posible ejecutar la aplicación en el emulador, seleccionando en la parte superior del proyecto el emulador a utilizar:

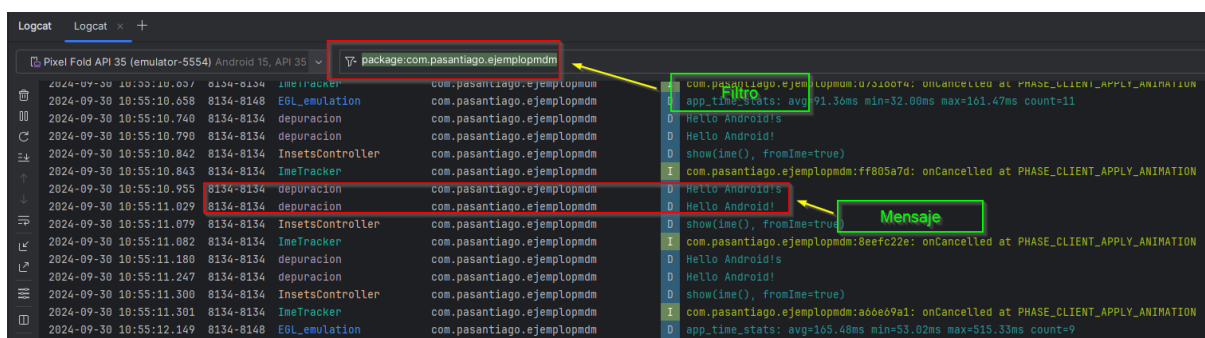


Es interesante y muy útil el log del dispositivo, **ya que no se tiene acceso al terminal para ver los clásicos “print”**. Existe una aplicación integrada llamada LogCat . Para enviar mensajes al log usar la clase Log (<https://developer.android.com/reference/android/util/Log>), estableciendo mensajes de tipo debug, info, warning, error entre otros.

Un ejemplo sencillo, de imprimir en el log depurando:

```
Log.d("Depuracion", "$it")
```

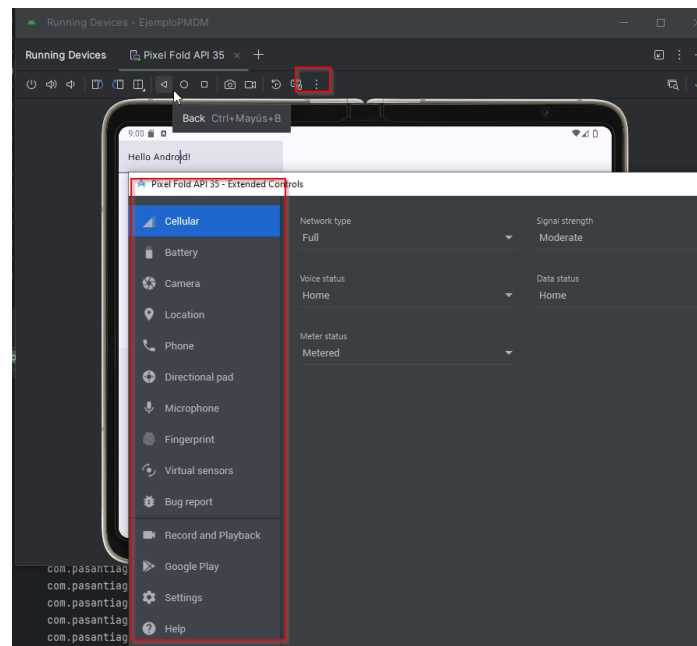
Pudiendo filtrar en LogCat, los mensajes en función de diferentes patrones:



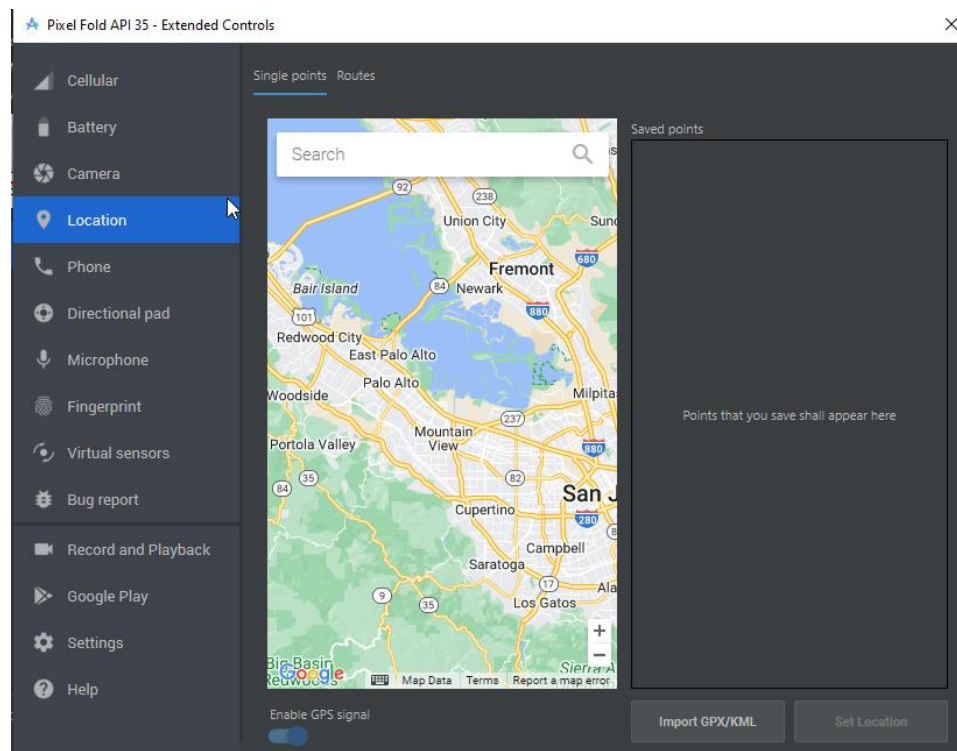
Existe un manual completo del manejo de LogCat de la documentación oficial:
<https://developer.android.com/studio/debug/logcat?hl=es-419>

6.1. Acceso al hardware del emulador.

De poco sirve un emulador de dispositivo móvil que no posee los mismos elementos que el dispositivo físico, para poder simular los dispositivos físicos (propios de cada tipo de dispositivo), simplemente pulsar en los 3 puntos verticales del emulador:



Y se abrirá una ventana para poder simular eventos hardware en el emulador:



7. Ejecutando en dispositivo real.

Es conveniente probar las aplicaciones en dispositivos reales, además en ocasiones los equipos de desarrollo no son capaces de utilizar con fluidez los emuladores, o posee un hardware especial.

Android Studio permite desarrollar en dispositivos hardware reales. Siendo necesario realizar previamente algunas acciones (extraído de la documentación oficial).

En el dispositivo, abrir la app de Configuración, seleccionar Opciones para desarrolladores y, luego, habilitar Depuración por USB (si corresponde).

- Windows: Instala un controlador USB para adb (si corresponde).
- Ubuntu Linux: Configura lo siguiente:

Cada usuario que quiera usar “adb” debe estar en el grupo “plugdev”. Si se muestra mensaje de error que indica que no está en el grupo plugdev, agrégalo con el siguiente comando:

```
sudo usermod -aG plugdev $LOGNAME
```

Los grupos solo se actualizan cuando se accede, por lo que se debe salir para que se aplique este cambio. El sistema debe tener instaladas reglas de udev que cubran el dispositivo. El paquete android-sdk-platform-tools-common contiene un conjunto predeterminado de reglas de udev mantenido por la comunidad para dispositivos Android. Para instalarlo, usa el siguiente comando:

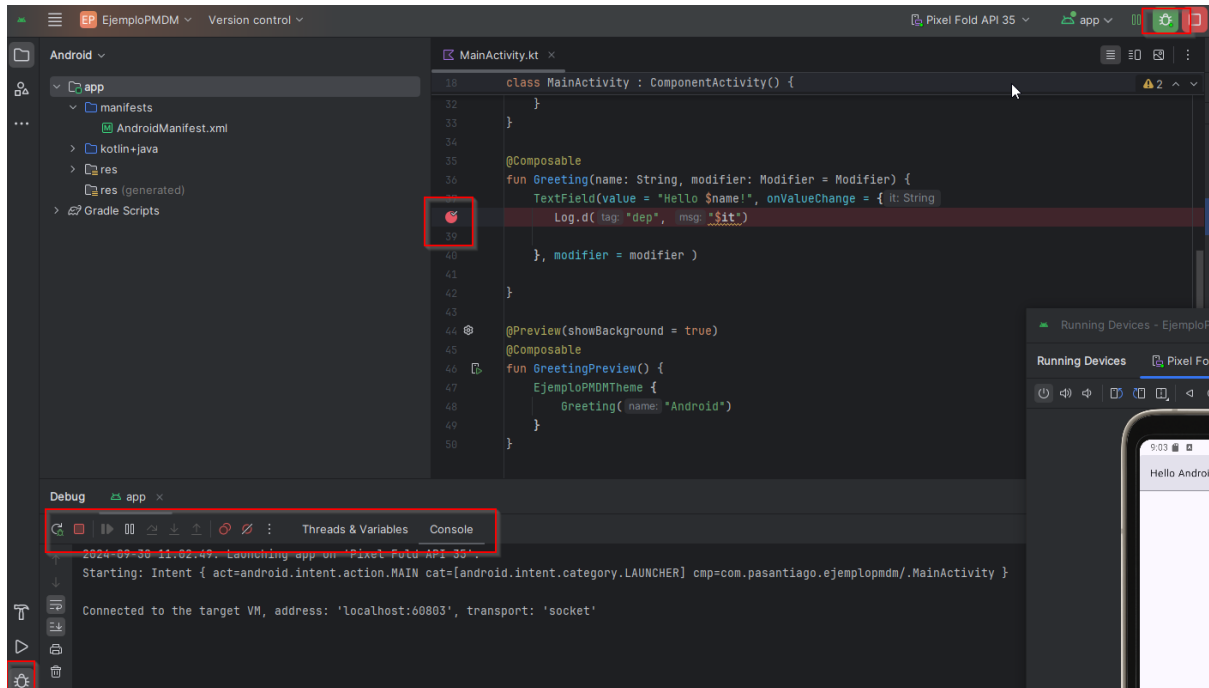
```
apt-get install android-sdk-platform-tools-common
```

Android 11 (y las versiones posteriores) admiten la implementación y depuración de aplicaciones de forma inalámbrica desde una estación de trabajo mediante Android Debug Bridge (adb). Por ejemplo, se puedes implementar una app depurable en varios dispositivos remotos sin conectar físicamente el dispositivo a través de USB y lidiar con problemas comunes de conexión USB, como la instalación de controladores.

8. Depurando.

Si se desea depurar aplicaciones en el emulador se procede de la misma forma que con cualquier otro IDE, marcar los puntos de rupturas y ejecutar en modo depuración

La depuración es una de las tareas más comunes en el desarrollo, Android Studio proporciona una herramienta que permite la comunicación con los dispositivos físicos, llamada Android Debug Bridge (adb).



9. Creando pruebas.

Las pruebas son una de las fases más importantes del desarrollo del software, en el caso de AndroidStudio se definen por defecto dos tipos de pruebas:

test (Pruebas unitarias):

- Ubicación: Directorio "src/test/java" en la estructura del proyecto.
- Propósito: Las pruebas unitarias se utilizan para probar unidades aisladas de código, como clases, métodos o funciones individuales, sin depender de la infraestructura de Android. Estas pruebas se ejecutan en el entorno de JVM (Máquina Virtual de Java) en lugar de en un dispositivo o emulador Android.
- Bibliotecas recomendadas: JUnit y Mockito son bibliotecas comunes utilizadas para escribir pruebas unitarias en Android.

androidTest (Pruebas instrumentadas):

- Ubicación: Directorio "src/androidTest/java" en la estructura del proyecto.
- Propósito: Las pruebas instrumentadas se utilizan para probar la interacción de la aplicación con el entorno de Android. Estas pruebas se ejecutan en un dispositivo real o emulador Android y pueden interactuar con las actividades, servicios y componentes de la interfaz de usuario de la aplicación.
- Bibliotecas recomendadas: Espresso y UI Automator son bibliotecas populares utilizadas para escribir pruebas instrumentadas en Android.

Para ejecutar las pruebas simplemente sobre el proyecto pulsar botón derecho e ir a las opciones "Run All Test", si se quiere ejecutar un test en concreto, realizar la misma acción, pero sobre el caso de prueba concreto.

The screenshot shows the Android Studio IDE with the following elements:

- Project View:** A context menu is open over the 'app' folder, showing options like 'New', 'Cut', 'Copy', 'Paste', 'Find Usages', 'Find in Files...', 'Replace in Files...', 'Refactor', 'Bookmarks', 'Reformat Code', 'Optimize Imports', 'Run \'All Tests\'', 'Debug \'All Tests\'', 'Run \'All Tests\' with Coverage', 'Modify Run Configuration...', 'Open In...', 'Local History', and 'Repair IDE on File'.
- Code Editor:** The file 'ExampleInstrumentedTest.kt' is open. It contains a Kotlin test class with the following code:


```
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.platform.app.InstrumentationRegistry
import org.junit.Assert.assertEquals
import org.junit.Test
import org.junit.runner.RunWith

/**
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals("com.pasantiago.ejemplopmdm", appContext.packageName)
    }
}
```
- Test Results:** A table shows the results of the tests. The table has columns for 'Test Results', 'Duration', and 'Pixel_Fold_API_35'.

Test Results	Duration	Pixel_Fold_API_35
✓ Test Results	45 ms	1/1
✓ ExampleInstrumentedTest	45 ms	1/1
✓ useAppContext	45 ms	✓
- Build Output:** The bottom panel shows the build output, indicating that the tests were successful.


```
Finished 1 tests on Pixel_Fold_API_35(AVD) - 15
Deprecated Gradle features were used in this build,
You can use '--warning-mode all' to show the individual
deprecations with the --warning-mode flag from the
command line.

For more on this, please refer to https://docs.gradle.org/7.4.2/userguide/command_line_interface.html#sec:command_line_warnings in the Gradle documentation.

BUILD SUCCESSFUL in 1m 1s
60 actionable tasks: 28 executed, 32 up-to-date
Build Analyzer results available
```

10. Actividades.

Documentar el proceso con comentarios y capturas de pantalla.

- 1) Descargar e instalar Android Studio.
- 2) Crear un nuevo proyecto con tus iniciales, por ejemplo, si tu nombre es Daenerys Targaryen Drago, el proyecto se llamará DTD, a partir de la plantilla “Empty Activity”, seleccionado el SDK Android 12, llamada “S”.
- 3) Buscar información sobre qué porcentaje del mercado podría soportar la aplicación creada.
- 4) Explicar las diferencias entre la vista “project” y la vista “Android”.
- 5) Crear un emulador para el proyecto anterior.
- 6) Ejecutar la aplicación en el emulador.
- 7) Modificar en el emulador el porcentaje de batería disponible y activar el micrófono.
- 8) Investigar como subir el código del proyecto a GitHub desde el IDE, el repositorio p1. se ha de llamar PMDM_TUS_INICIALES_p1.
- 9) Crear la clase Persona en Kotlin:

```
class Persona {  
    // Atributos privados  
    private var nombre: String  
    private var edad: Int  
  
    // Constructor primario  
    constructor(nombre: String, edad: Int) {  
        this.nombre = nombre  
        this.edad = edad  
    }  
  
    // Getter para obtener el nombre  
    fun getNombre(): String {  
        return nombre  
    }  
  
    // Setter para modificar el nombre  
    fun setNombre(nuevoNombre: String) {  
        nombre = nuevoNombre  
    }  
  
    // Getter para obtener la edad  
    fun getEdad(): Int {  
        return edad  
    }  
}
```

```
// Setter para modificar la edad
fun setEdad(nuevaEdad: Int) {
    edad = nuevaEdad
}

fun saludar() {
    println("Hola, mi nombre es $nombre y tengo $edad años.")
}
}
```

10) Crear un caso de prueba en test que compare 2 personas en el fichero ExampleUnitTest llamado iguales_TUS_INICIALES. Para crear dos instancias de persona en Kotlin:

```
var p1: Persona = Persona(nombre="Paco", edad = 34);
var p2: Persona = Persona(nombre="Paco", edad = 34);
```

11) Ejecutar el test. ¿Qué sucede? ¿Cómo solucionarlo?

12) Sobre escribir el método necesario para que el test anterior funcione.

Opcional:

Configurar un dispositivo real y ejecutar el proyecto creado.