ÍNDICE

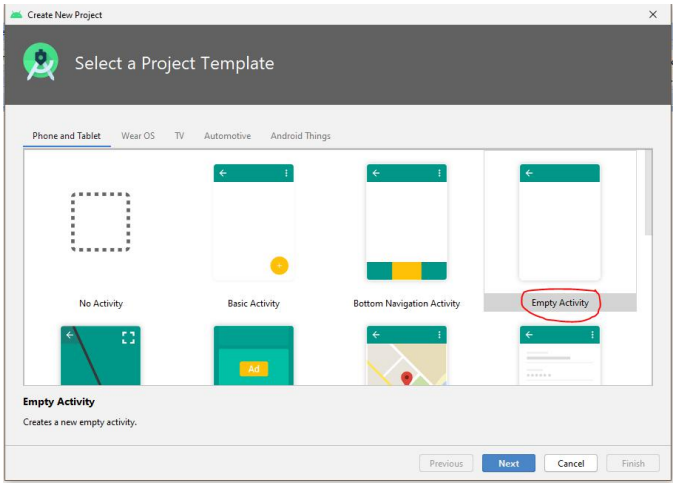
1.1. Introducción

Empezaremos por crear un proyecto que servirá de base para desarrollar el curso.

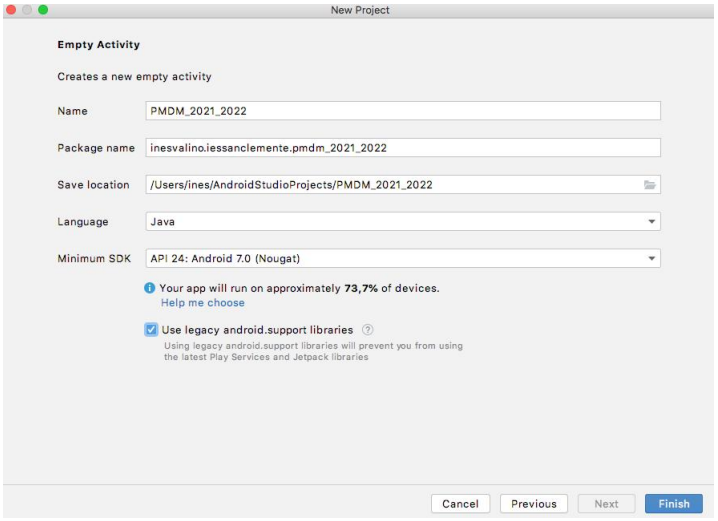
Si el estudiante quiere puede crear un proyecto diferente para cada ejercicio o para cada unidad ...

1.2. Crear un proyecto base

- En cada unidad crearemos un paquete diferente dentro del mismo proyecto.
  En cada paquete pondremos las diferentes actividades creadas.
- Crea un nuevo proyecto:
  - Indicamos que desarrollaremos una aplicación móvil.
  - Elegimos uno de los patrones. En el ejemplo será una pantalla en blanco.



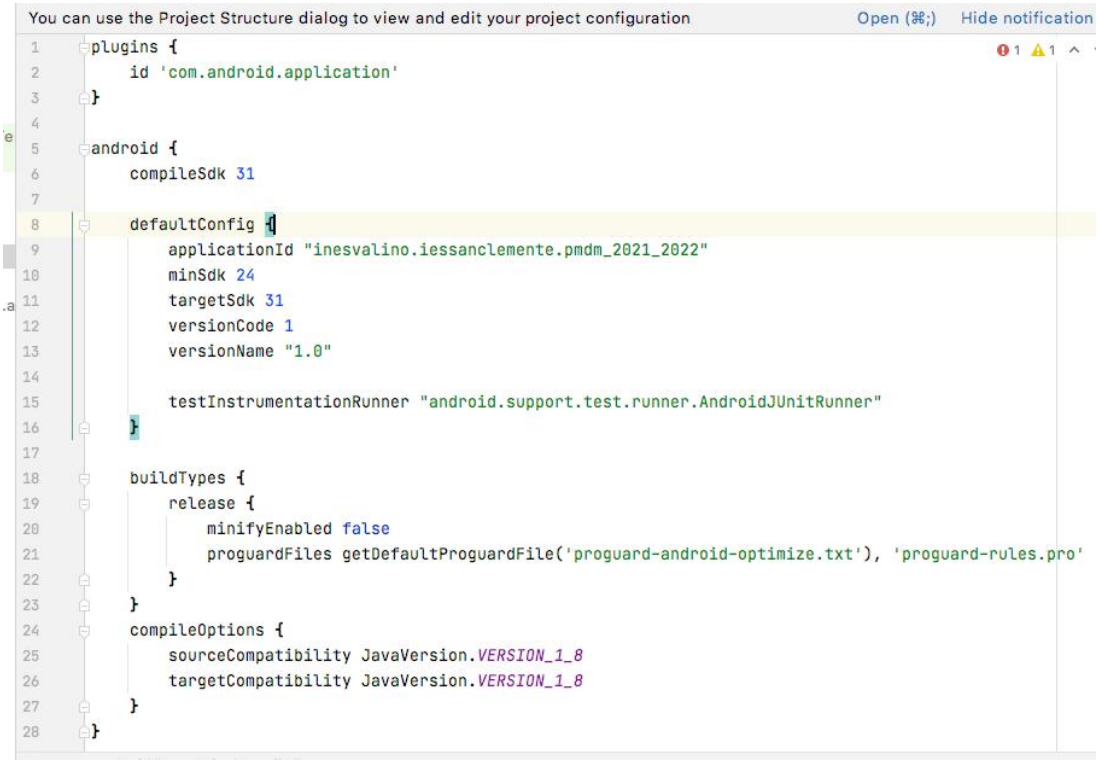
- Nombre de la aplicación: **PMDM_2021_2022**.
- Nombre del paquete: ***nombreprimerapellidoalumno*.iessanclemente.pmdm_2021_2022** (no escribir acentos, o ñ en el nombre y primer apellido)
- Carpeta donde se guarda el proyecto: Podemos dejar lo que trae por defecto.
- Idioma: **Java**
- Nivel mínimo de API: 24
- Marcaremos la opción *Use legacy android.support libraries*



Pulsar el botón **Finalizar**.

- Datos del archivo **build.gradle** a nivel de módulo (puede ser que tengáis otras versiones dependiendo de la versión de Android Studio instalada):
  - MinSDK: 24: Es lo que indicamos cuando creamos el proyecto.
  - Target_SDK: 31
  - Compile_SDK: 31

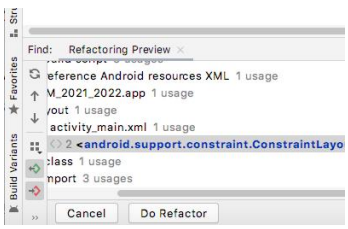
Nota 1: Explicaremos el significado en un apartado posterior.



Nota 2: Es posible que te aparezca en la parte inferior de la ventana de Android Studio, el siguiente problema:

Cuya descripción completa es: "*Version 28 (intended for Android Pie and below) is the last version of the legacy support library, so we recommend that you migrate to AndroidX libraries when using Android Q and moving forward. The IDE can help with this: Refactor > Migrate to AndroidX...*"

Para resolverlo: ve a la barra superior y clica sobre "Refactor", ahí elige la opción "Migrate to AndroidX", y clica en "Migrate". Luego pulsa el botón "Do Refactor" que encontrarás en la parte de abajo del IDE:

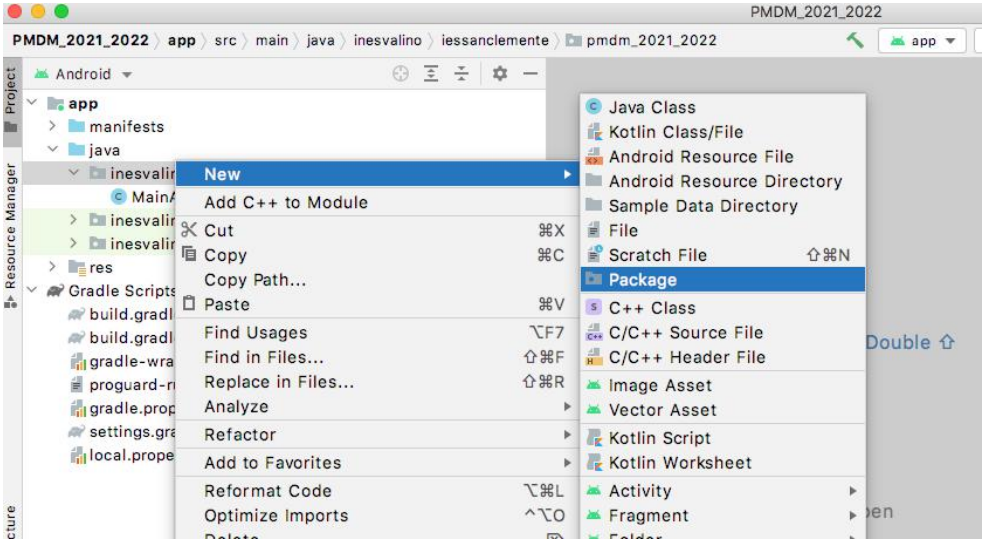


De esta forma, Android Studio modificará automáticamente todos los archivos necesarios para migrar usando las librerías de AndroidX

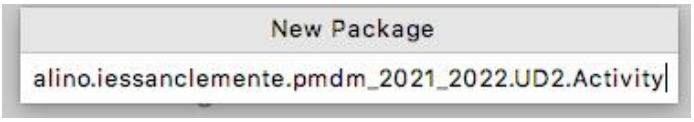
```
androidTestImplementation 'androidx.test.ext:junit:1.1.1'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.0'
```

A lo largo del curso, se crearán diferentes paquetes (vienen como subdirectorios). Veamos un ejemplo de cómo crear un nuevo paquete ejemplo llamado '**UD2.Activity'**. Recordad dejar al menos una actividad en el paquete inicial, porque si no gráficamente, Android Studio la hace desaparecer.

Creando un nuevo paquete



Hacer clic derecho sobre el paquete en el que queremos crear un subpaquete.



Teclear el nombre del paquete a crear.

Ahora, al hacer clic derecho en el nuevo paquete, podemos crear nuevas actividades que se incluirán en él.

En cualquier momento podemos mover actividades de un paquete a otro (arrastrando con el ratón) pero al hacerlo aparecerá una ventana en la que tendremos que elegir la opción **Refactorizar**.

- Aunque no se indica en las distintas actividades creadas, cada vez que se cree una actividad tipo ***Launcher*** (actividad de lanzador) hay que modifica el archivo **AndroidManifiest.xml** para darle una etiqueta a cada una de ellas. Los datos de la etiqueta serán los mismos que el nombre de la actividad. Hacemos esto para poder identificar dentro del emulador / dispositivo si vamos a la pantalla donde ves todas las aplicaciones instaladas:

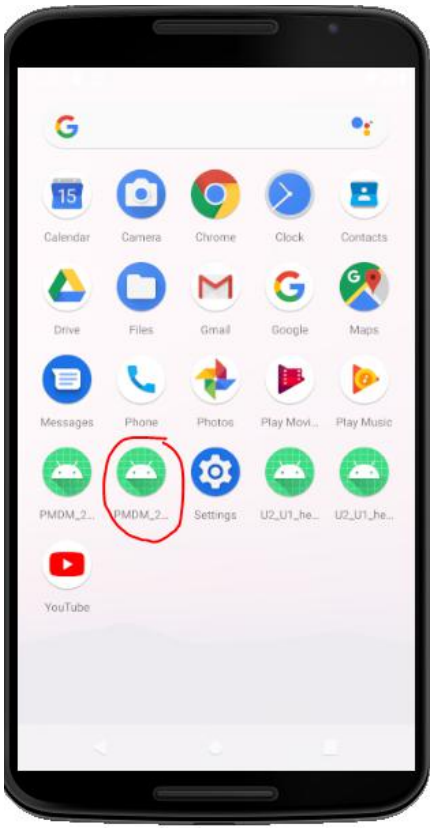
```
<activity android:name=".UD2.Activity.UD02_01_ConstraintLayout">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

En este caso, la actividad está en un paquete de nombre **UD2.Activity**, pero el nombre de la actividad es: **UD02_01_ConstraintLayout.**

Nota: En la clase de la nueva actividad (el código .java) podría dar un error similar a "*error: package R does not exist en android studio"* en tal caso simplemente añade: import inesvalino.iessanclemente.pmdm_2021_2022.R;

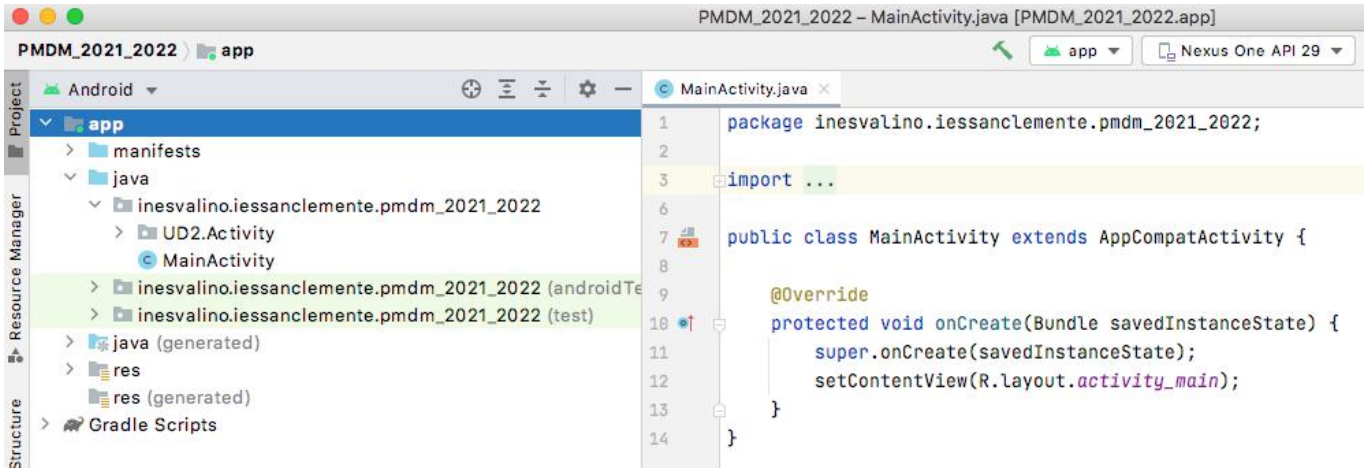
Si ejecutamos esta aplicación:

Arrastramos el ratón desde abajo para mostrar las aplicaciones instaladas en el emulador.



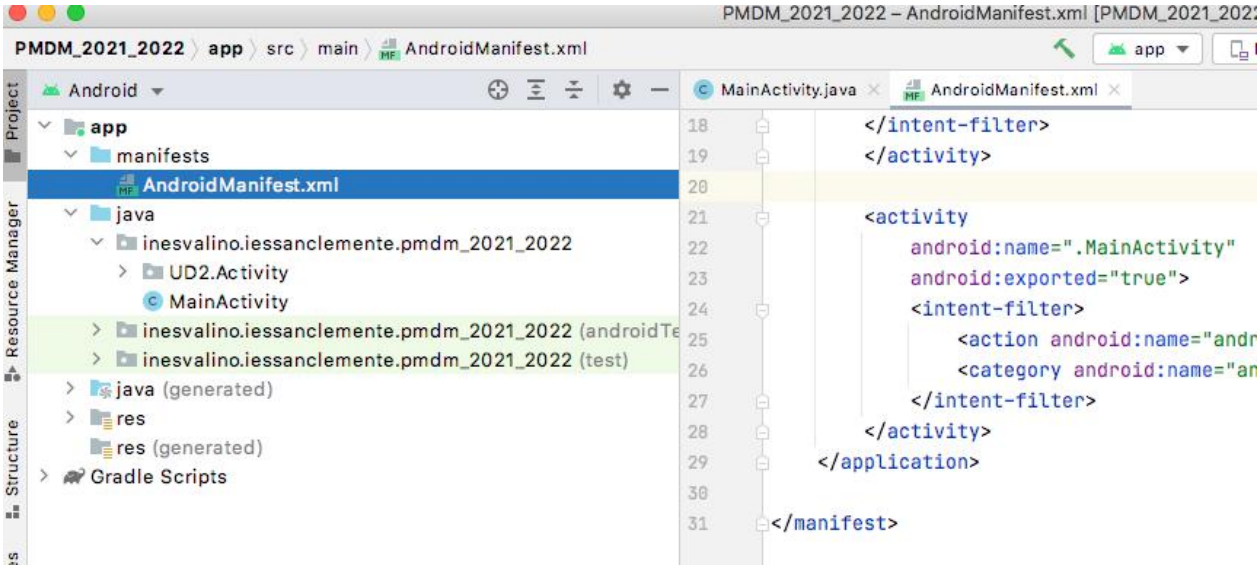
Aparece la aplicación instalada.

1.3. Nombre de las clases y layouts

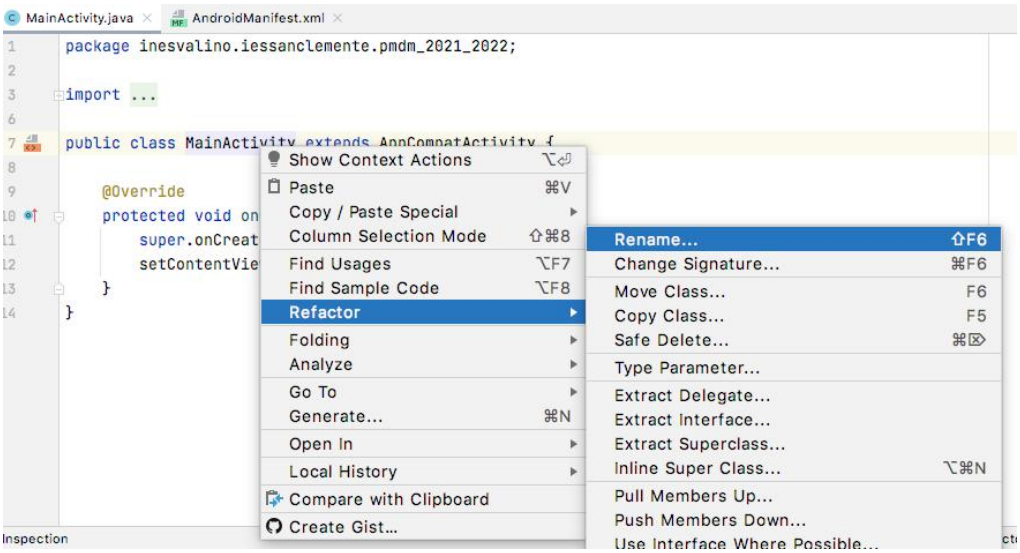


Como en Java, el nombre de la clase pública debe coincidir con el nombre físico del archivo (en el ejemplo ilustrado por la imagen de arriba el nombre es MainActivity).

Como vemos, la clase se deriva de la clase *AppCompactActivity*, no de la clase Activity. A lo largo de este curso, puede que veamos ejemplos de código en los que las actividades se derivaban de la clase Activity (en versiones anteriores del IDE). Ignoraremos esto y haremos que todas las actividades se deriven de AppCompactActivity (recordad cambiarlo!!!).

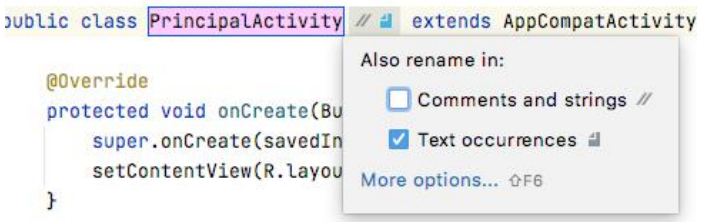


Si queremos cambiar el nombre no basta con cambiar el nombre de la clase como se usa en otros sitios, como *AndroidManifiest.*xml (tal y como ilustra la imagen anterior). La forma de cambiar el nombre en todos lugares donde sea necesario es la siguiente:

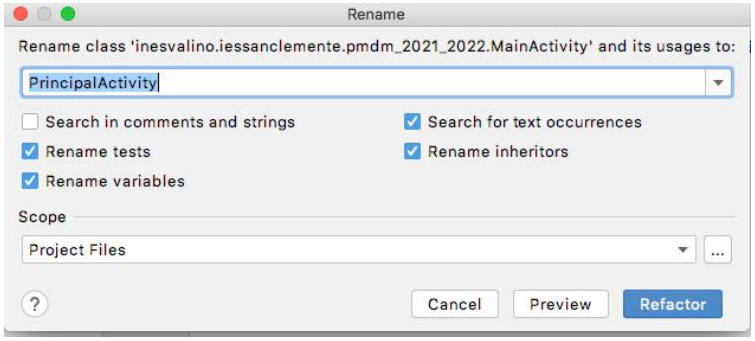


Haced clic derecho sobre lo que queremos cambiar (por ejemplo MainActivity) y elegid la opción **Refactor => Renombrar**.

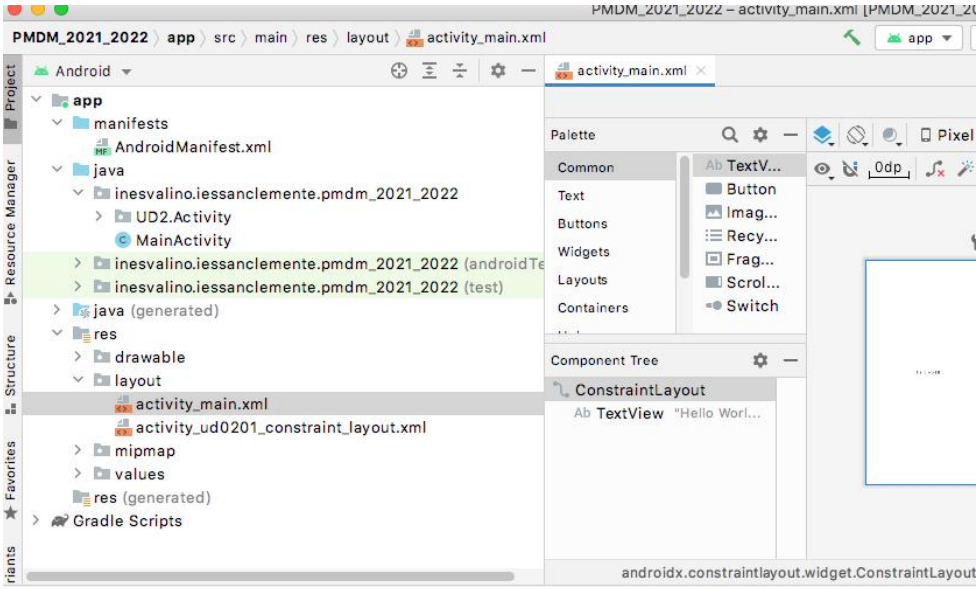
Al hacerlo os dará la opción de cambiarlo incluso en comentarios o strings,



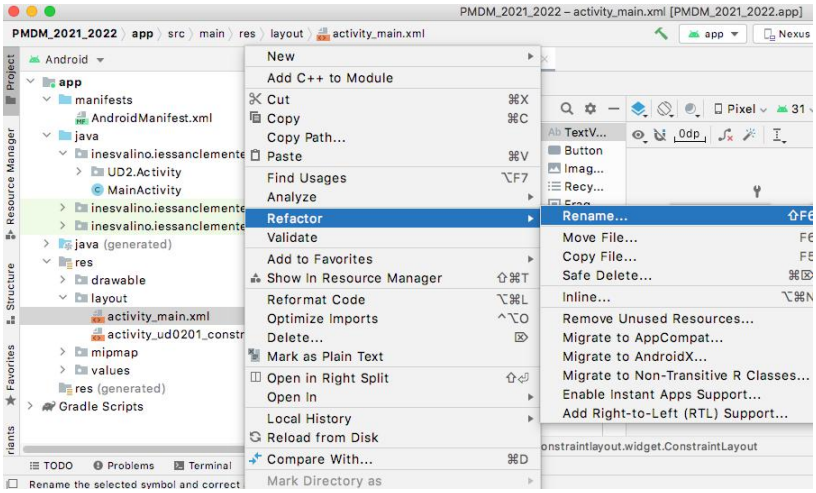
y si hacéis clic sobre *More options*, os da la opción de elegir en cambiarlo a nivel de archivos de Proyecto o en todos los lugares, mi consejo es que ante la duda, elegid en todos los lugares para evitaros haber olvidado cambiarlo en algún sitio.



El **Layout** es donde se encuentran los componentes gráficos de la Activity siguiendo una distribución. Podemos acceder presionando la tecla Control y haciendo clic con el mouse sobre el nombre del diseño...



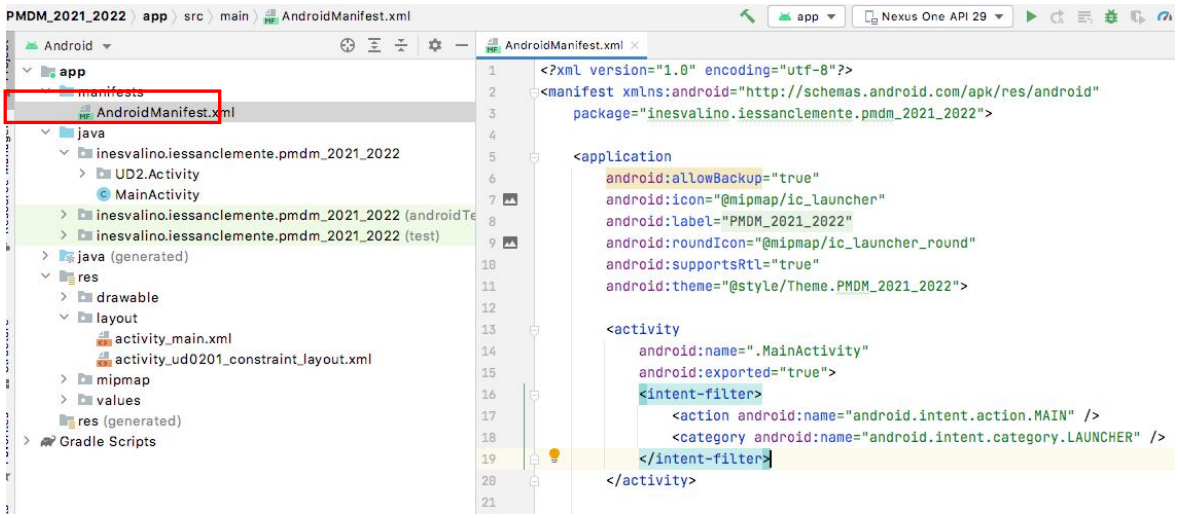
O podemos ir a la carpeta de **res => layout** y hacer doble clic en ella.



Para cambiar el nombre haremos lo mismo que en la clase.

1.4. Archivo AndroidManifest.xml

Como hemos comentado anteriormente, este archivo contiene la definición de las características más importantes de la aplicación: nombre, versión, icono, actividades, servicios, instrumentación, permisos de acceso a recursos, etc.



- En la imagen podemos ver entre otros datos:
  - Nombre del ícono a cargar (se refiere a un ícono ubicado en **/app/res/mipmaps/**
- Nombre de la aplicación que se define en una constante en el archivo **/res/values/strings.xml**. Es el nombre que aparece asociado al icono de la aplicación.
- Cómo se verá la aplicación (es similar a un archivo CSS HTML). Se encuentra en el archivo **/res/values/styles.xml** o **/res/values/themes/themes.xml.** Podemos crearlos y personalizarlos a nuestro gusto: https://developer.android.com/guide/topics/ui/look-and-feel/themes?hl=es-419
- Las actividades que componen el proyecto. Inicialmente solo tenemos una.

1.5. Identificar el minSDK, targetSDK e buildSDK

- Anteriormente en Eclipse, dentro del archivo **AndroidManifest.xml** también estaban las líneas que indicaban el targetSDK y el minSDK.
- Como se mencionó anteriormente, la API determinará la versión del sistema operativo en el que desarrollaremos la aplicación.

  Cada nueva versión incorpora nuevos componentes gráficos y nuevas funcionalidades, pero tenemos que mantener un equilibrio entre la versión a elegir y el mercado que queremos monopolizar.
- La API también determinará qué aplicaciones se pueden instalar en el dispositivo móvil, ya que una aplicación desarrollada para una versión superior no se puede instalar en un sistema operativo Android con una versión inferior.

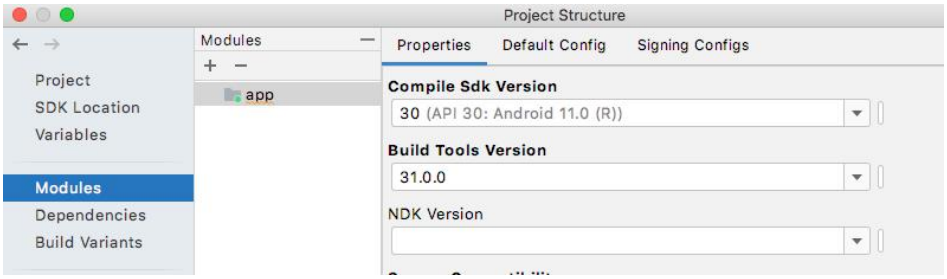
- Al determinar la API en un proyecto de Android debemos elegir lo siguiente:
  - **MIN SDK**: Será la API mínima (versión del SO Android) en la que se puede ejecutar nuestra aplicación. No se puede instalar en una versión anterior.
  - **TARGET SDK**: Indica en qué versión del sistema operativo Android se prueba nuestra aplicación y sabemos que funciona correctamente. Suele ser el más alto. Podría funcionar en una versión posterior o inferior (siempre que sea más grande que minSdk). El uso de un objetivo u otro puede afectar la apariencia de nuestra aplicación en tiempo de ejecución (por ejemplo, puede cambiar el tema o la apariencia predeterminados, según el objetivo)
  - **BUILD SDK o Compile SDK**: Indica la versión del SDK que usamos para compilar nuestra aplicación. Esto limitará los recursos que podemos usar en el desarrollo de la aplicación, porque si por ejemplo, tenemos una nueva característica en una API por ejemplo, en la API 22, y hemos marcado un BUILD_SDK de 19, no podemos usarla. El build_sdk siempre tendrá que ser mayor o igual que min_sdk.

Generalmente se sigue la regla:

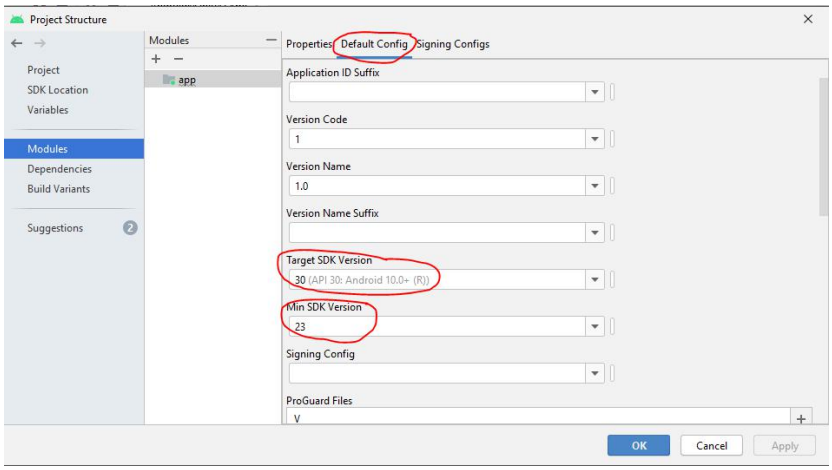
minSdkVersion (mínimo posible) <= targetSdkVersion == compileSdkVersion (SDK más alto posible)

Para especificar la API de un proyecto Android, clicamos sobre su icono en la barra de herramientas

o bien, en la barra de tareas escogemos la opción **File -> Project Structure** (o con la combinación de teclas indicada).



Debemos elegir el módulo **App** y en la pestaña **Properties** tenemos el BUILD SDK. Debemos asegurarnos de que también se seleccione un **build-tools**, generalmente la última versión.



En la pestaña **Default Config** tenemos a MIN SDK y TARGET SDK.

Estos cambios se verán reflejados en el archivo **build.gradle** a nivel de módulo.

Nota: Estos datos se pueden definir en el archivo AndroidManifiest.xml, pero si están en dos sitios, tendrá preferencia (sobrescribe) el que se escribe en el archivo **build.gradle**.