MI PRIMER PROYECTO CON ANDROID STUDIO

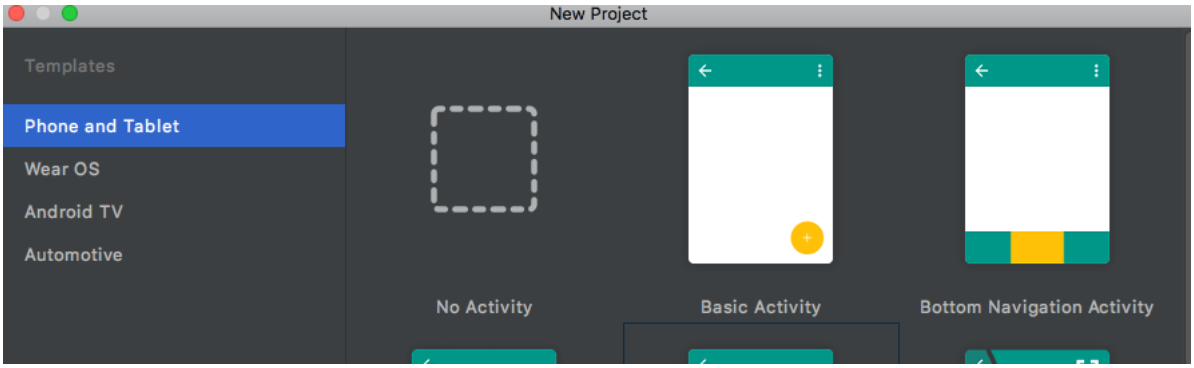
Utilizar un entorno de desarrollo nos facilita mucho la creación de programas, sobre todo en el caso de Android dado que se tendrán que usar una gran variedad de programas. Android Studio permite crear y gestionar proyectos de forma muy rápida, acelerando los ciclos de desarrollo.

1.1 Creación de un primer proyecto

Para crear un primer proyecto Android con Android Studio sigue los siguientes pasos:

En la pantalla de bienvenida del IDE selecciona ***New Project***

A continuación podrás elegir la plataforma para la que queremos desarrollar (Teléfonos y tabletas, Wear OS, TV, ..) y el tipo de actividad inicial que quieres en tu aplicación. En este módulo nos centraremos en las aplicaciones para teléfonos y tabletas.



¿Qué es una Actividad (Activity)?

Una Activity es de lo más básico y a su vez más utilizado a la hora de desarrollar nuestras aplicaciones para Android.

Una actividad es nuestro programa en sí mismo, contiene la interfaz de usuario de nuestra App. Sirven como punto de entrada par ala interacción del usuario con una app, y también son fundamentales para que el usuario navegue en la app o entre diferentes apps.

Podemos decir que cada pantalla de una aplicación es una Activity. Una aplicación estará compuesta por una o varias Activities.

Una Activity puede ser destruida por la propia aplicación al pulsar el botón Back del dispositivo por el sistema porque está en la pila de aplicaciones en segundo plano y se necesita los recursos que está consumiendo

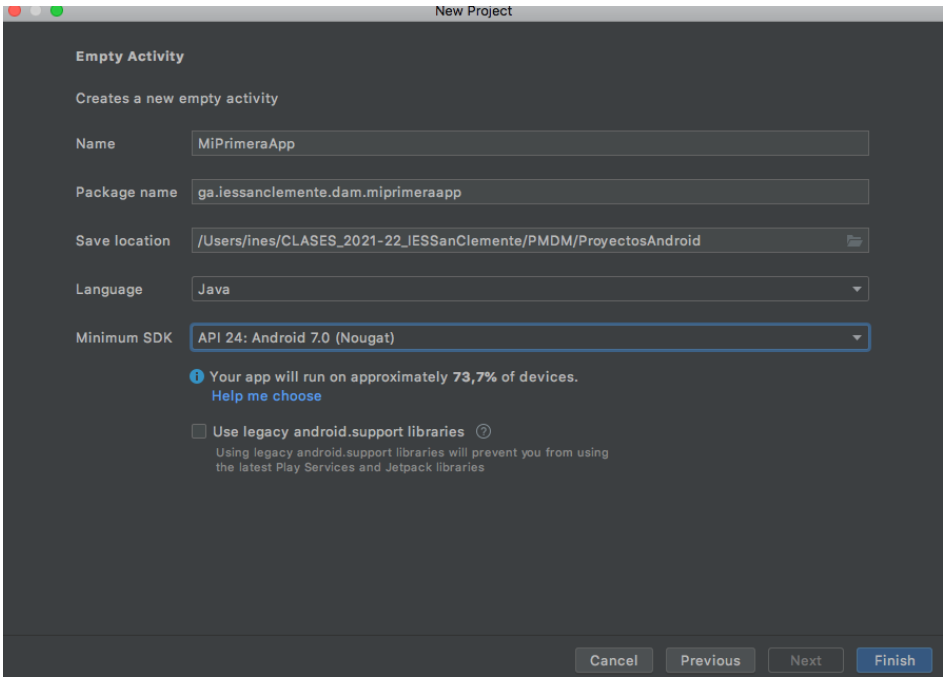
Como veremos más tarde en detalle, las Activities están formadas por dos partes:

- la parte lógica: archivo .java que es la clase que se crea para poder manipular, interactuar y colocar el código de esa actividad

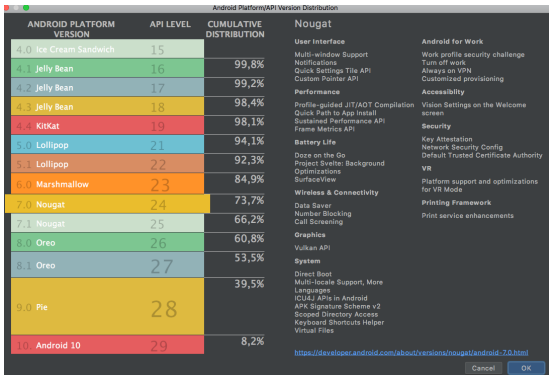
- la parte gráfica: es un xml que tiene todos los elementos que estamos viendo de una pantalla.

Por lo tanto, una actividad o activity está formada por la parte lógica (archivo JAVA) y la parte gráfica (archivo XML).

A continuación escogemos el tipo de actividad inicial, en este caso una actividad "en blanco" ***Empty Activity***. Pulsa ***Next*** para para pasar a la pantalla donde se rellenan los detalles del proyecto:



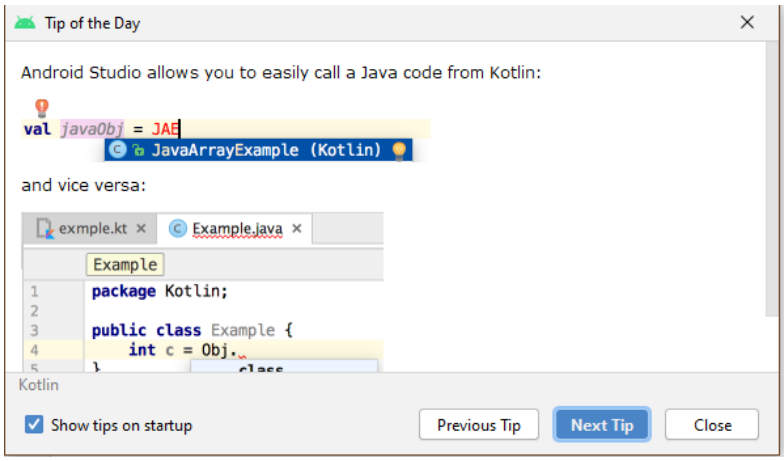
- Name: es el nombre que le daremos a nuestra aplicación
- Package name: es el nombre bajo el que queremos que se muestren el paquete java de la aplicación. Las clases Java que creemos pertenecerán a este paquete. Incluye el dominio de la compañía. En este ejemplo es ga.sanclemente.dam.*miprimeraapp*
- Save location: ubicación donde se almacenarán todos los ficheros del proyecto.
- Language: Escogemos el lenguaje con el que quieres programar (selecciona Java).
- Minimun SDK: versión mínima del API para la que desarrollamos nuestra app. Por lo tanto, la app no podrá ser instalada en dispositivos con una versión inferior. La decisión depende de los requisitos de nuestra aplicación, pero para empezar, hay que escoger una que sea bastante compatible con todos los móviles y tabletas que hay en la actualidad. Para ayudarte a tomar esta decisión puedes clicar sobre la opción "Help me to choose" donde se muestra un gráfico con el porcentaje de usuarios que podrán instalar la aplicación de acuerdo a su nivel de API y un resumen con las nuevas características introducidas en este nivel de API.



Por ejemplo la API 24: Android 7.0 (Nougat) es compatible con aproximadamente el 74% de dispositivos

Una vez has seleccionado has completado la configuración pulsa ***Finish*** y comenzará el proceso de creación del proyecto y configuración del entorno para que empieces a programar (si es la primera vez, le puede llevar unos minutos).

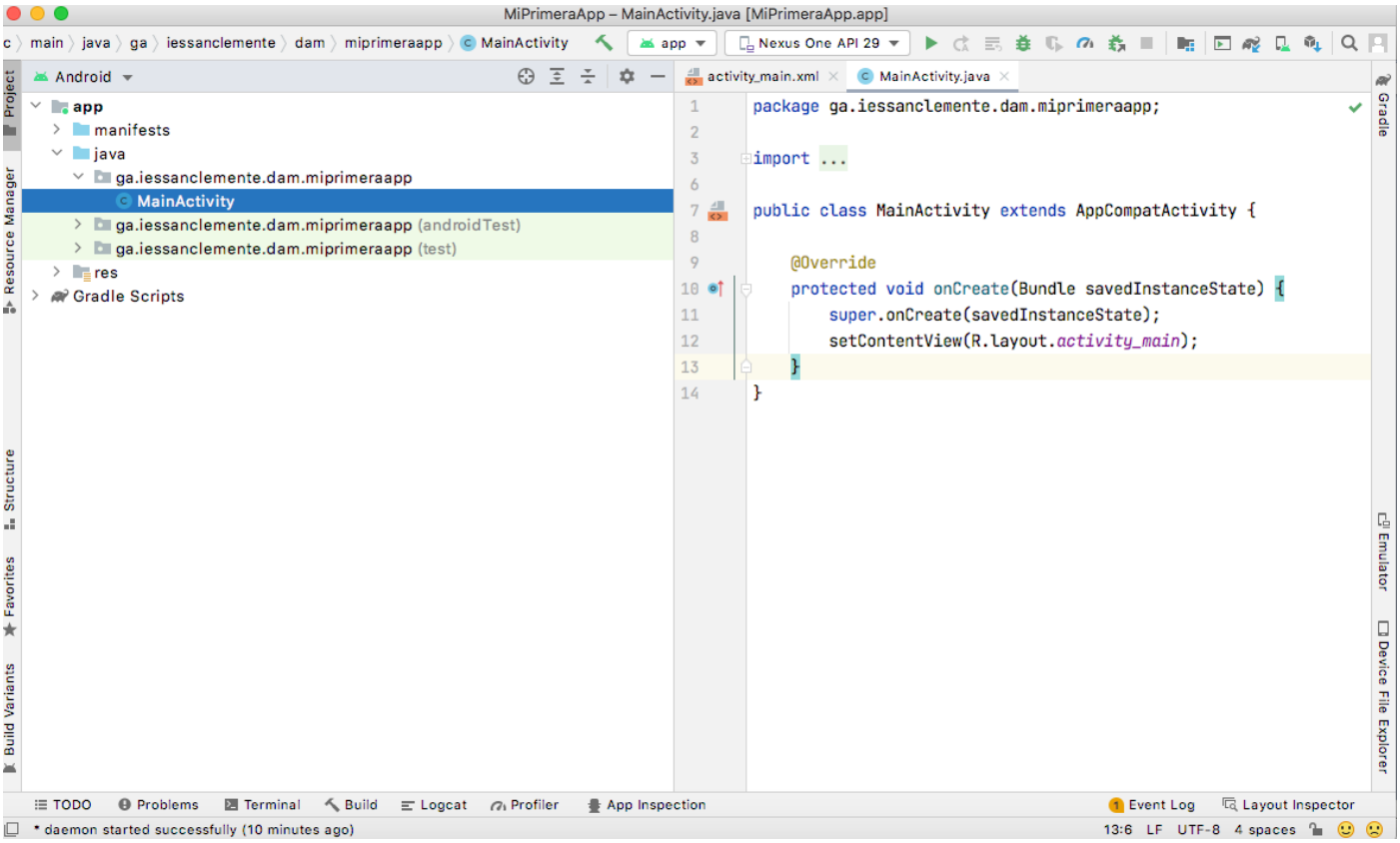
Una vez creado el proyecto, Android Studio nos da la bienvenida amablemente con consejos sobre cómo utilizarlo:



Es importante leerse estos consejos porque a la larga facilitan el aprendizaje del uso de la herramienta y pueden proporcionar trucos para efectuar operaciones que a priori pueden parecer no triviales.

1.2 Interfaz de usuario

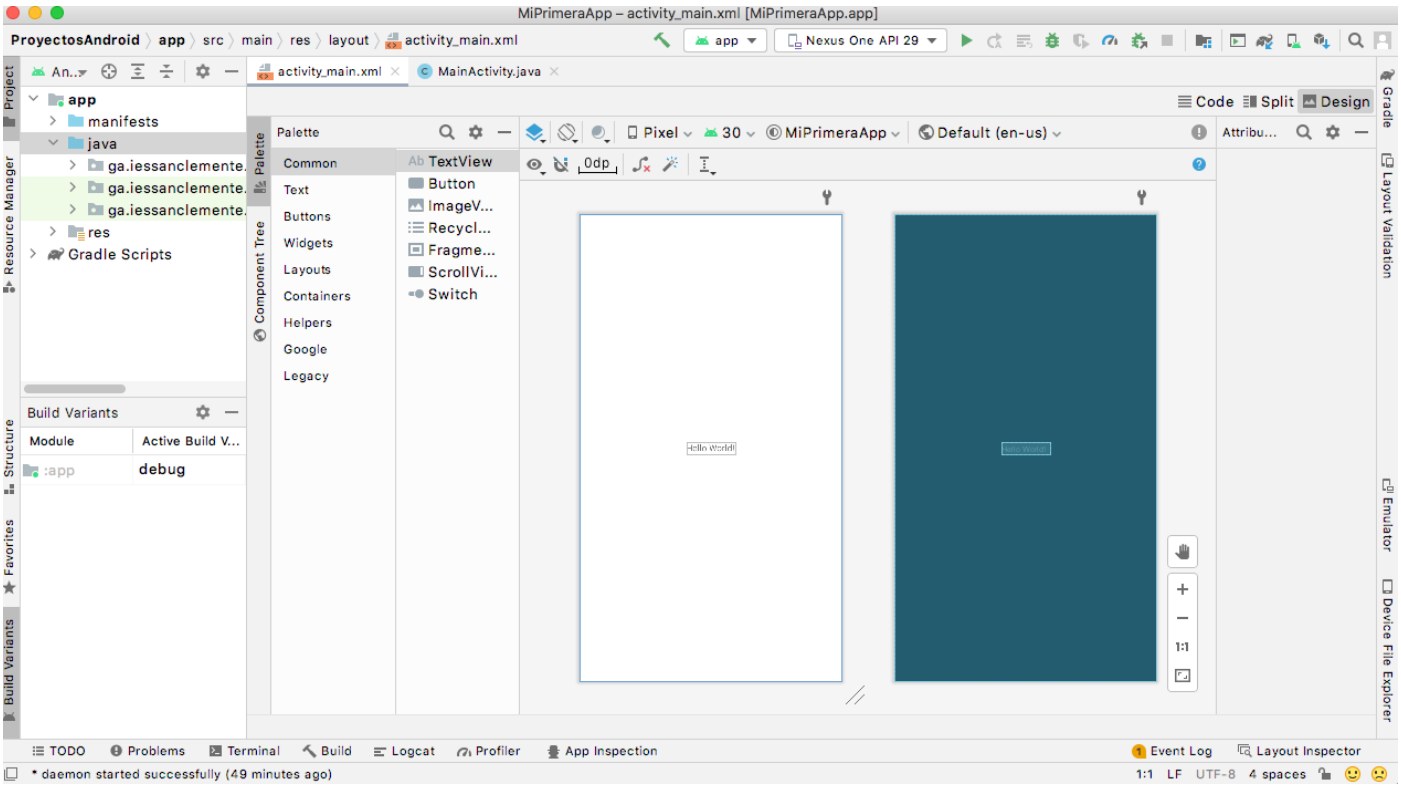
Una vez abierto el proyecto aparecerá esta pantalla:



Esta ventana principal consta de varias áreas lógicas, entre ellas:

- En la parte superior, la **barra de herramientas y de navegación** te permite realizar una gran variedad de acciones, como ejecutar tu app e iniciar las herramientas de Android.
- A la izquierda, el **explorador de proyectos**, a la izquierda, donde aparecen todas las carpetas y ficheros que compone el proyecto. Los modos más útiles de visualización son modo **Android** (como imagen de ejemplo) y modo **Project**.
- El **panel central**, es el área de edición para crear y modificar código, y dependiendo que estemos editando se mostrará de una forma diferente:
  - "MainActivity.java" verás código Java. Éste es el código Java que se genera automaticamente cuando creas el proyecto y donde se define el comportamiento de la actividad.
  - "activity_main.xml" verás el archivo de diseño.
  -

Si seleccionas el editor de diseño el panel central se verá de la siguiente manera (depende también de lo que tengas seleccionado en la barra que está a la derecha del todo: aquí Designer-Atributtes) :



Puedes ver tres pestañas o tabs en la parte superior derecha, *'Design'*, *'Code' y 'Split'*. Con la vista *'Design'* tendrás una visión de todos los componentes o *Widgets*, que puedes ir insertando para configurar tu interfaz gráfica. Con la vista Con la vista *'Code'* verás que aparece con una ventana con código XML autogenerado por Android Studio.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

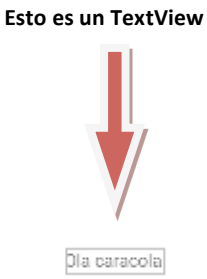
La interfaz gráfica de tu app se puede y se recomienda definir con definiciones en XML. Si te fijas en los *tabs* superiores, por un lado, tienes el fichero XML y por otro lado, un fichero Java. Pues en XML se declaran todos los componentes y en el fichero Java se programan los comportamientos. Y por último, la vista '*Split'* una mezcla de las anteriores.

En modo '*Split'* con el código xml a la izquierda, y la vista previa a la derecha puedes ver como quedará tu App en el dispositivo Android. Si modificas el archivo XML verás cómo cambia. Puedes experimentar a cambiar alguna cadena de texto para ver cómo cambia el diseño. Por ejemplo, el fragmento de código:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    />
```

Corresponde a un campo de texto que se presenta en pantalla y que contiene la cadena de texto '*Hello World*'. Si pruebas a cambiar la cadena de caracteres por '*Ola Caracola*', verás el efecto en el emulador.

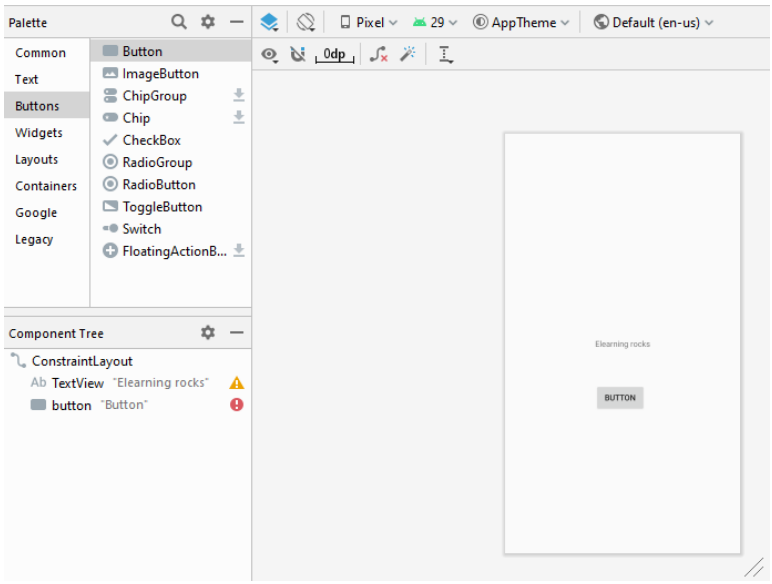
```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Ola caracola"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



1.3 Programando (diseñando) sin escribir código

Una de las características del desarrollo de Android es que se puede diseñar muchas cosas sin apenas tocar código, de hecho, ya habrás comprobado la cantidad de código en XML y Java que genera Android Studio con solo tan solo trastear un poco con los menús.

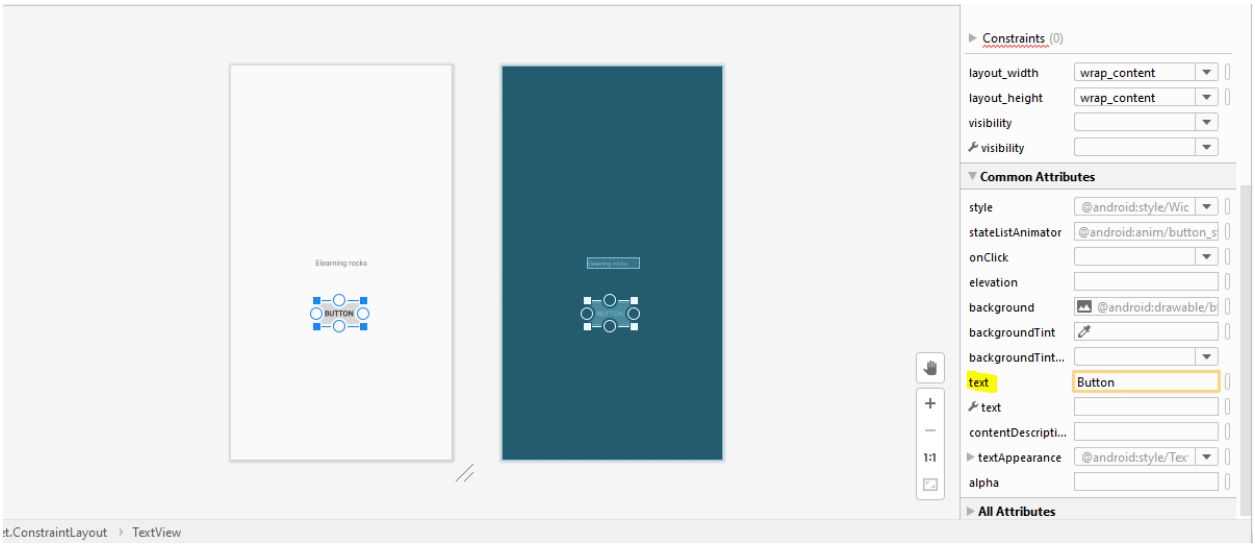
Para ilustrarlo, haz la siguiente prueba: En vista diseño, arrastra un *widget* de tipo botón (*Button*) a la pantalla del móvil que muestra la vista previa y sitúalo justo debajo del *TextView* que viene por defecto.



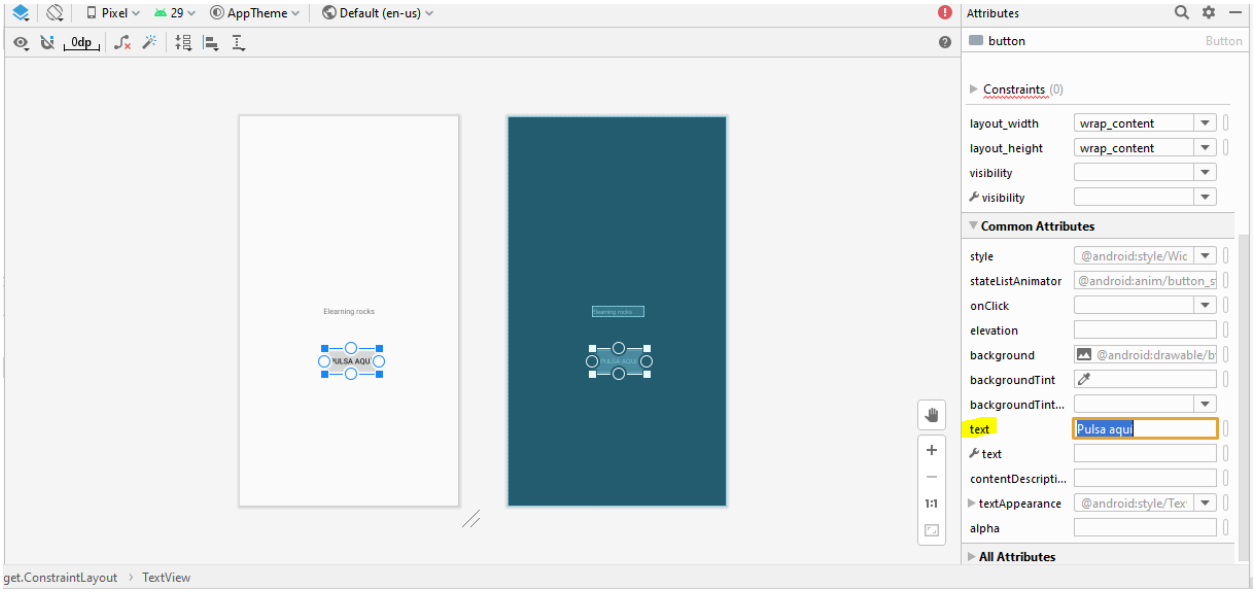
Si cambias a la vista de texto "*Code*", verás que se ha añadido en XML el siguiente código:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    tools:layout_editor_absoluteX="158dp"
    tools:layout_editor_absoluteY="436dp" />
```

Para cambiar el texto del botón puedes hacerlo editando el código pulsando dos veces sobre el propio botón y abriendo las propiedades:



Puedes ponerle en el campo *text* una cadena de caracteres con el nuevo texto del botón y el id lo usarás para referenciarlo después desde del código. Puede ponerle, por ejemplo, "***Pulsa aquí***".



1.4 Introduciendo un poco de código

El siguiente paso es darle funcionalidad a nuestro primer proyecto. En concreto que cuando pulsemos el botón, cambie el texto del TextView.

Lo primero de todo que debes saber es que para poder referenciar en tu código a las clases de componentes que has incluido en el XML y que van a ser parte de tu interfaz de usuario, debes importar las clases. Dentro del paquete Android, subpaquete widget tienes las clases *TextView* y *Button*, que son las dos que has agregado en tu primer proyecto y que aparecen en el código Java:.

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
```

Para referenciar en el código Java los componentes que has agregado mediante el código XML de la actividad solo tienes que crear una referencia al objeto de la clase que quieras, por ejemplo botón (*Button*) y llamar a la función *findViewById*(…)

```
Button miBoton;
miBoton=(Button) findViewById (R.id.button);
```

A partir de aquí puedes acceder a un sinfín de propiedades y métodos para programar tu botón como te apetezca.

Lo siguiente es saber dónde ubicar tu código. Si buscas una función *main*, que sepas que no lo vas a encontrar. De hecho, no solo no existe como tal, sino que cada actividad tiene un ciclo de vida, que va sucediendo llamadas a funciones callback según la actividad experimente una interacción por parte del usuario, por ejemplo, arrancar una actividad, abandonar una actividad, retomar una actividad. A continuación puedes ver el gráfico extraído de la página de desarrolladores de Android, que ilustra perfectamente el ciclo de vida de una actividad y la transición de llamadas a funciones callback según van pasando por diferentes estados:

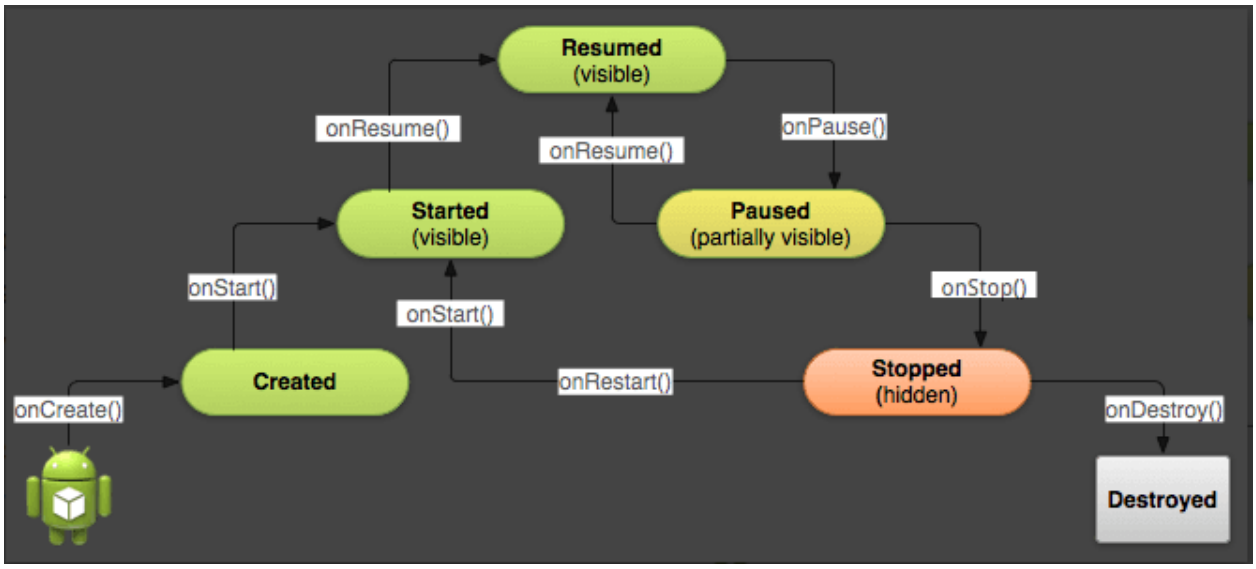


Imagen: Ciclo de vida de una actividad de developer.android.com

De esta manera, con la arquitectura de actividad de Android se asegura que nuestra aplicación será una app adaptada a un dispositivo móvil y no un programa típico para un ordenador de tipo Desktop, es decir, se evita:

- Que la actividad se bloquee o deje de funcionar cuando el usuario recibe una llamada o cambio a otra app mientras está usando la tuya.
- Que consuma recursos valiosos del sistema cuando el usuario no está usándola activamente.
- Que se pierda el progreso del usuario si abandonan la app y luego vuelve a ella.
- Que se bloquee cuando el usuario, por ejemplo, cambia la posición de la pantalla de vertical a horizontal.
- Etc.

No es necesario implementar todas las funciones callback, aunque conforme tus apps sean más completas y más complejas, tendrás que usar muchas de ellas.

De momento necesitas centrarte en la primera acción que el ciclo de vida ejecuta cuando el sistema operativo arranca la App que estás desarrollando. Esta función callback es "***onCreate***", similar a la función main, pero con diferencias técnicas.

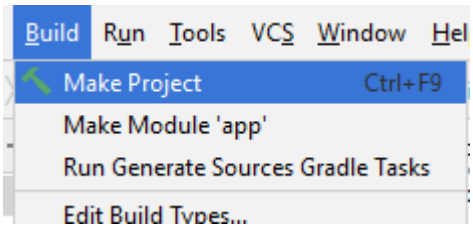
En primer lugar has de añadir la implementación de la clase *View.OnClickListener*, método "de escucha" que se llama cuando el usuario en este caso toca el elemento botón. Después añade el código para acceder a los widgets (button y textView) que has agregado en el fichero XML y finalmente conseguir acceso a ellos. Después añade en el código la función *onCreate* el código para poder referenciar a los componentes textView y button, y registra el listener '*OnClick'*. A continuación se muestran las líneas de código que hemos añadido:

```
public class MainActivity extends AppCompatActivity {
    Button miBoton;
    TextView miTexto;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        miBoton = (Button) findViewById(R.id.button);
        miBoton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                miTexto = (TextView) findViewById(R.id.textView);
                miTexto.setText("pulsando");
            }
        });
    }
```

Y a compilar…..



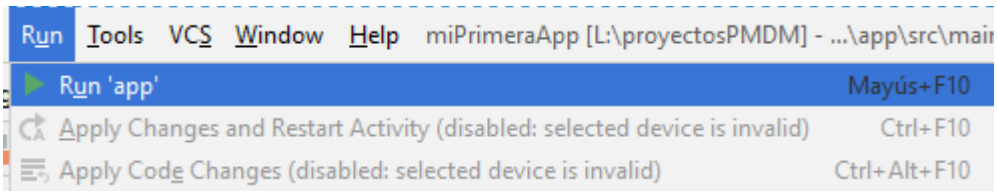
Si tuvieras errores saldría algo de este estilo:

Donde nos indica que en la línea 22 hay un error, en este caso, no encuentra una variable de nombre 'textview' y efectivamente así es. Comprobamos en en fichero XML, que el 'id' del textView es 'tex-tView' con V mayúscula, recordad que java es sensitive-case, es decir deferencia entre mayúsculas y minúsculas.

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Elearning rocks"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

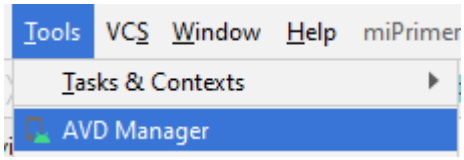
Corregimos el error y volvemos a compilar…

Ejecutamos...

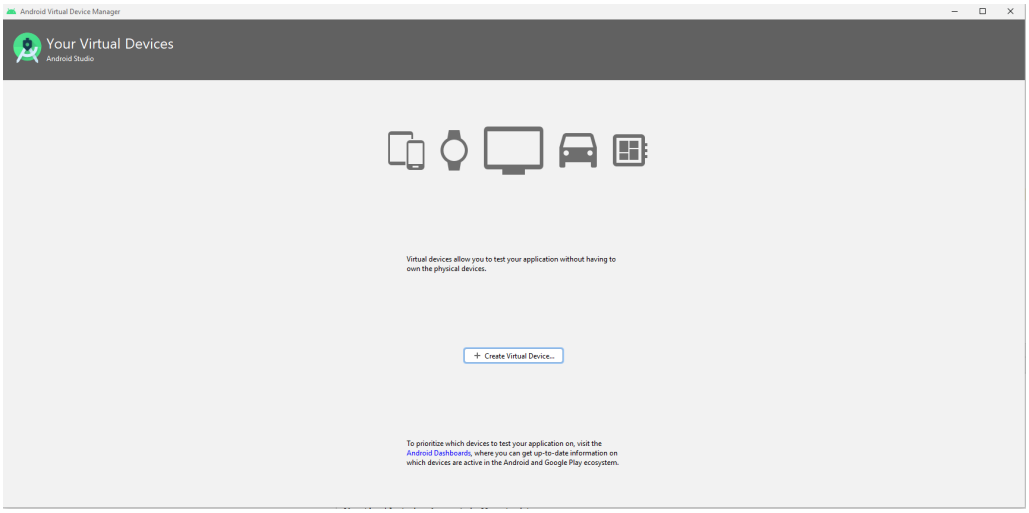


Y....¡anda! ¡No tenemos ejecutando el emulador! No hay problema.

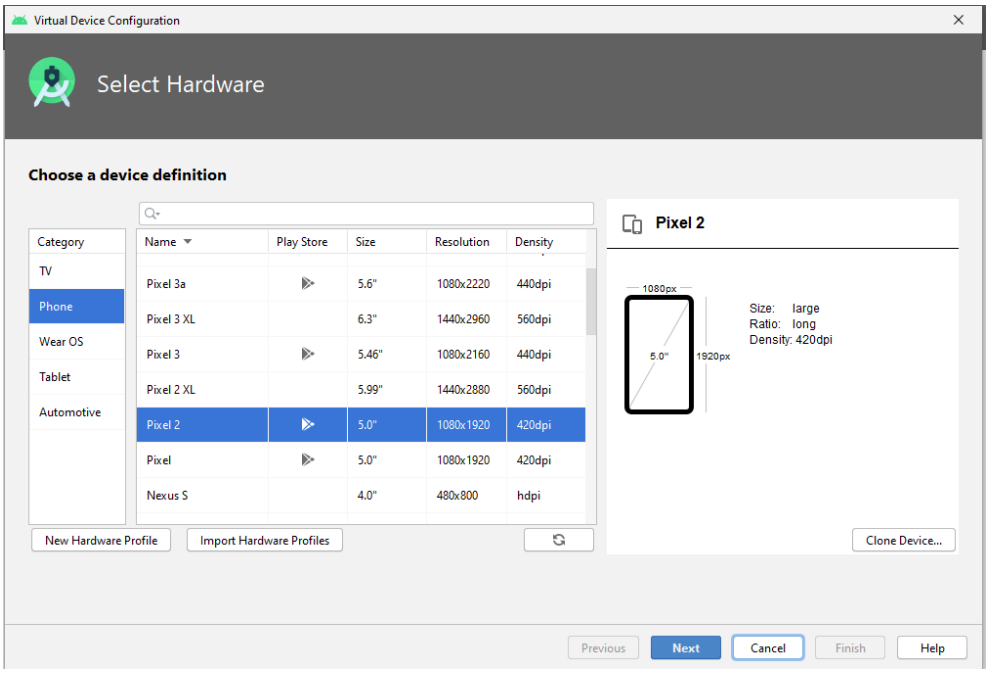
Es hora de crear el emulador, el teléfono virtual donde poder ejecutar y depurar nuestros programas. Para crearlo, sacamos el Android Virtual Device Manager (**AVD Manager**).

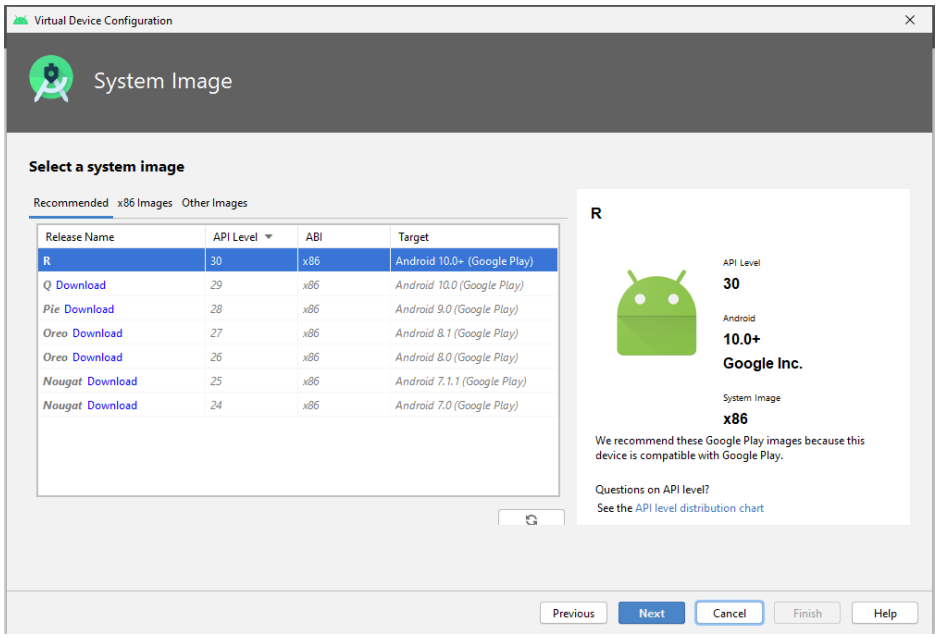


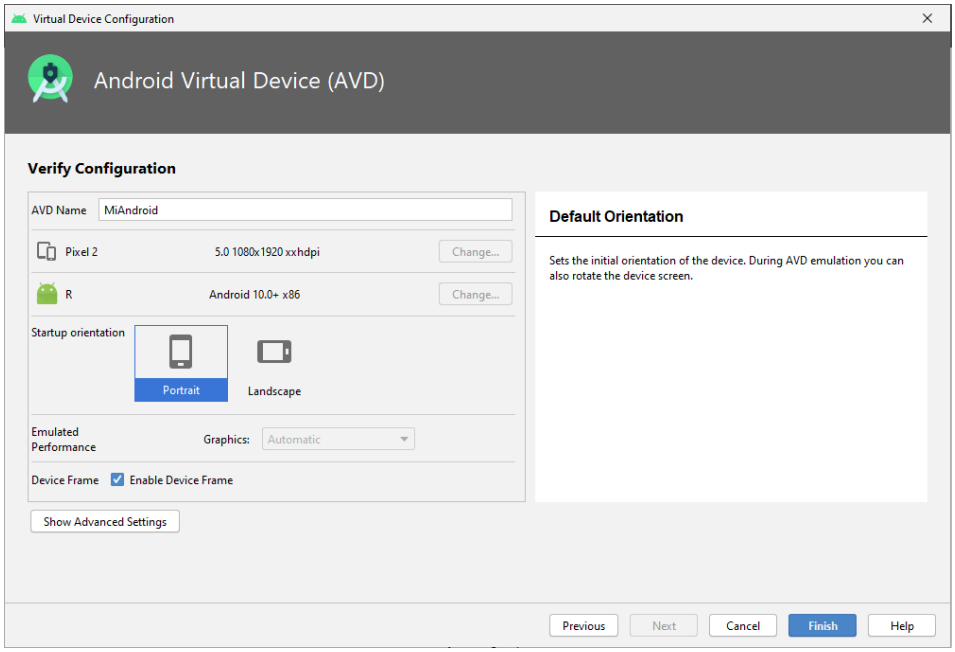
El AVD Manager es muy sencillo de usar:

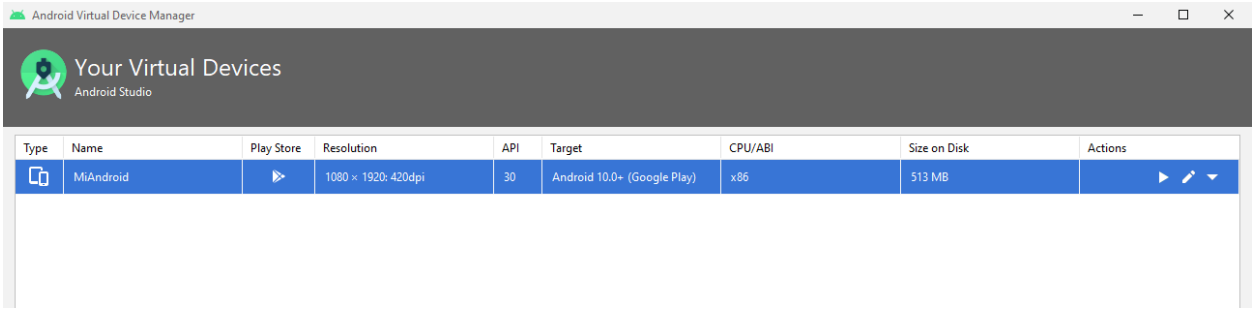


Podemos crear tantos dispositivos virtuales como queremos, de momento solo nos hace falta uno:

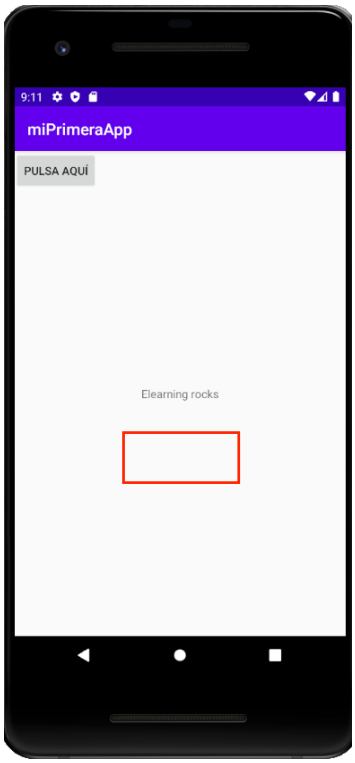



Virtual Device Configuration
System Image
Select a system image
Recommended x86 Images Other Images
Release Name API Level ABI Target
R 30 x86 Android 10.0+ (Google Play)
Q Download 29 x86 Android 10.0 (Google Play)
Pie Download 28 x86 Android 9.0 (Google Play)
Oreo Download 27 x86 Android 8.1 (Google Play)
Oreo Download 26 x86 Android 8.0 (Google Play)
Nougat Download 25 x86 Android 7.1.1 (Google Play)
Nougat Download 24 x86 Android 7.0 (Google Play)
R
API Level
30
Android
10.0+
Google Inc.
System Image
x86
We recommend these Google Play images because this device is compatible with Google Play.
Questions on API level?
See the API level distribution chart
Previous Next Cancel Finish Help

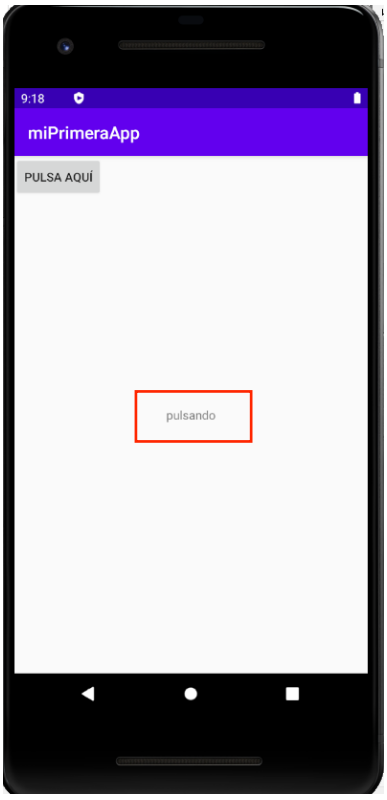

Virtual Device Configuration
Android Virtual Device (AVD)
Verify Configuration
AVD Name MiAndroid
Pixel 2 5.0 1080x1920 xxhdpi Change...
R Android 10.0+ x86 Change...
Startup orientation
Portrait
Landscape
Emulated Performance
Graphics: Automatic
Device Frame Enable Device Frame
Show Advanced Settings
Default Orientation
Sets the initial orientation of the device. During AVD emulation you can also rotate the device screen.
Previous Next Cancel Finish Help


Android Virtual Device Manager
Your Virtual Devices
Android Studio
Type Name Play Store Resolution API Target CPU/ABI Size on Disk Actions
MiAndroid 1080 × 1920: 420dpi 30 Android 10.0+ (Google Play) x86 513 MB

1.5 Probando …



Ahora sí, compilado el código y creado el emulador, volvemos a lanzar la ejecución de app y esta vez, podemos seleccionar el dispositivo creado. Puede tardar un poco en arrancar, pero una vez arrancado no es necesario arrancarlo de nuevo entre ejecución y ejecución de tu app.



1.6 Entendiendo un poco más el código

Varios aspectos fundamentales deben quedarte claro desde este ejemplo:

A. La necesidad de conseguir una referencia a los widget de la interfaz de usuario.

```
Button miBoton;
miBoton = (Button) findViewById(R.id.button);
```

La primera instrucción, declara la referencia, la segunda, consigue el acceso al widget y a partir de ahí, ya podemos operar con él. Ten en cuenta que este código debe ser situado después de la instrucción `setContentView(R.layout.activity_main);` Si no lo haces, el resultado de la llamada a `findViewById()` será nulo y no podrás acceder al widget.

B. El registro de la función callback:

```
miBoton.setOnClickListener(new View.OnClickListener()
```

Esta es la referencia al objeto creado de la clase actual, que como implementa la función de callback *OnClick*, pues se puede pasar como parámetro.

C. La programación de la función OnClick():

```
public void onClick(View view) {
    miTexto = (TextView) findViewById(R.id.textView);
    miTexto.setText("pulsando");
}
```

Consistente en obtener la referencia al objeto de texto y establecer el valor pulsado (método *setText*).