

Introducción

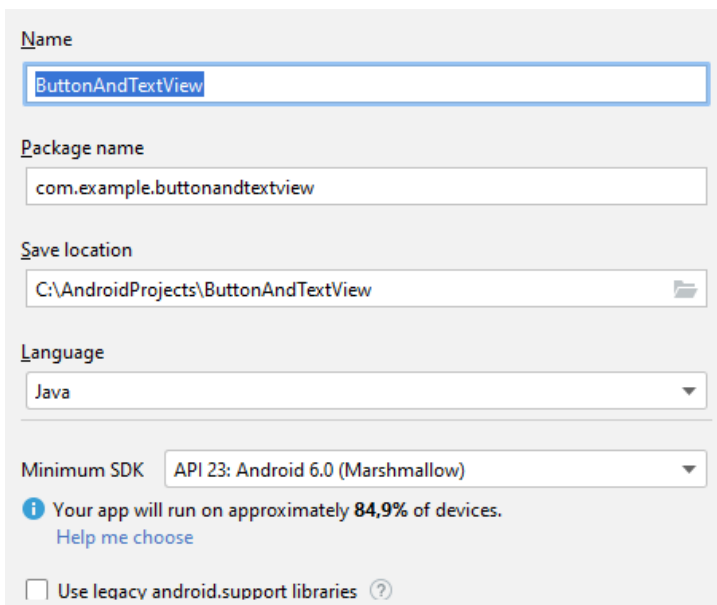
Antes de empezar en serio a programar con el Android Estudio, ver lo que es una *activity*, un *widget*, o cómo se le añade un icono a la app; vamos a hacer una primera práctica totalmente guiada. **El objetivo** es implementar un sencillo Hola Mundo en Android e instalarlo en un dispositivo móvil (o emularlo). La intención es hacernos a la idea de cómo se trabaja con el Android Studio antes de entrar en materia.

Se pide: Una App en Android que disponga de un **Button** y un **TextView**. El **Button** tiene que estar etiquetado como “Ok”. El **TextView** mostrará el texto “Pulsa el botón”. Cuando el usuario presiona dicho botón, el **TextView** mostrará el texto “Hola Mundo”. Si el usuario presione el botón de nuevo, el **TextView** mostrará otra vez “Pulsa el botón”.

SOLUC: P0 - ButtonAndTextView.zip

Primer paso – Nuevo proyecto

En el Android Studio, hacemos un nuevo proyecto con **Create New Project**. En la ventana de **Project Template** se nos ofrecen diferentes plantillas para crear nuestras Apps, para facilitarnos el desarrollo. Podemos escoger la que queramos, pero para este ejemplo escogeremos la **Empty Activity**.



The screenshot shows the 'Create New Project' dialog in Android Studio. The fields are filled as follows:

- Name:** ButtonAndTextView
- Package name:** com.example.buttonandtextview
- Save location:** C:\AndroidProjects\ButtonAndTextView
- Language:** Java
- Minimum SDK:** API 23: Android 6.0 (Marshmallow)

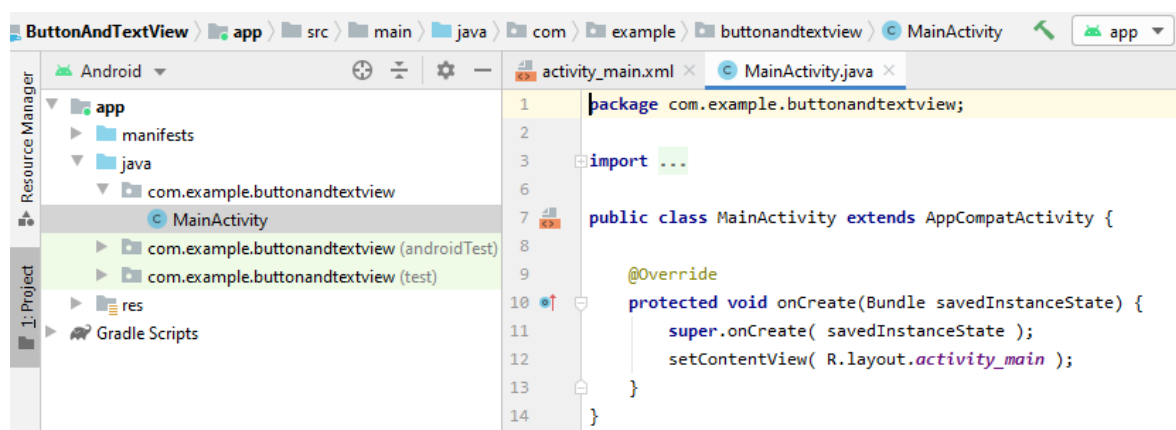
Below the fields, there is an information icon and text: "Your app will run on approximately 84,9% of devices. Help me choose". At the bottom, there is a checkbox labeled "Use legacy android.support libraries" with a question mark icon.

Llamaremos al proyecto **ButtonAndTextView**. Fíjate en que se crea automáticamente un paquete con el nombre que le has dado. En este paquete irán las clases Java del proyecto, de forma similar a un proyecto Java convencional. Por el momento no importa la API de Android que usemos, así que puedes poner la Marshmallow si quieres. Es posible cambiarlo más adelante.

Recuerda que es importante tener clara la Versión de Android con la que vas a trabajar, pero por el momento basta con una que os funcione en vuestros dispositivos (y Java).

Segundo paso – La interfaz

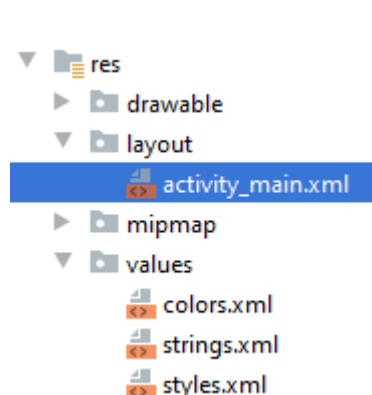
Una vez inicializado el proyecto, veremos una ventana similar a esta. Por defecto, se nos ha creado un montón de carpetas, de ficheros, en el árbol de directorios; y se nos abren por defecto dos ficheros: **MainActivity.java** y **activity_main.xml**.



Ambos ficheros **MainActivity.java** y **activity_main.xml** definen *una actividad* de nuestra app. Concretamente, en el **activity_main.xml** se define cómo es visualmente la app, y en el **MainActivity.java** cómo se comportan los componentes de esa parte visual. Dicho de otra forma, en el **activity_main.xml** ‘colocaremos’ el **Button** y el **TextView**, y en el **MainActivity.java** diremos ‘qué pasa’ cuando pulsamos el botón (eventos).

Vamos primero al **activity_main.xml**

Este fichero se encuentra en **app/res/layout/activity_main.xml**.

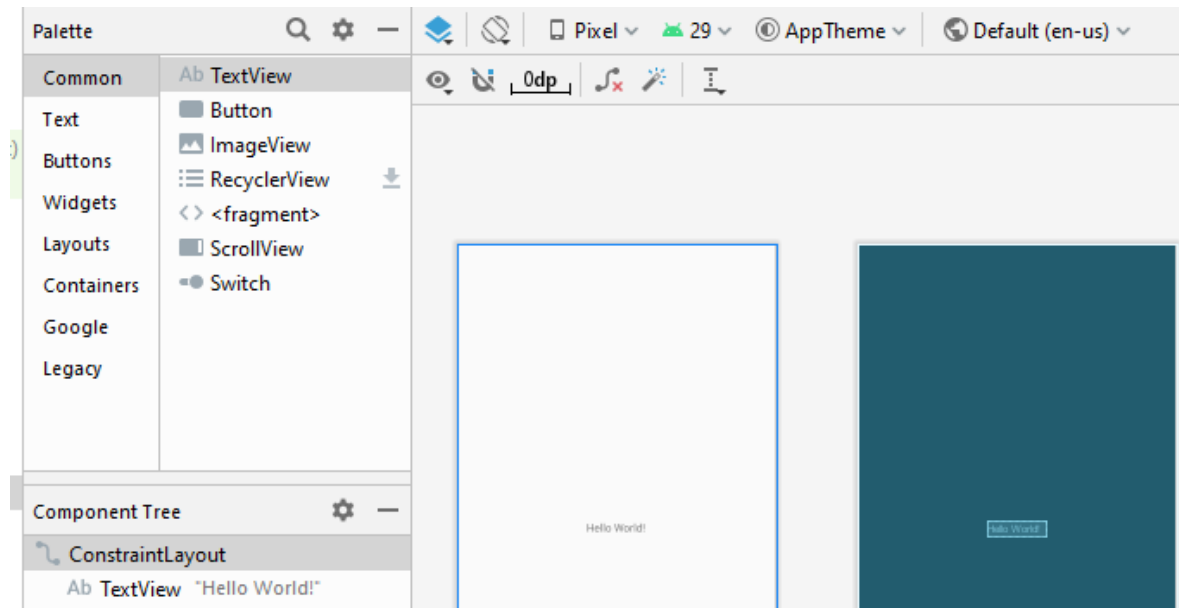


En la carpeta **res** es donde se colocan todos los recursos de la app. Imágenes, sonidos, textos... Si te fijas, podrás hacerte una idea de qué hacen varios ficheros:

- **Strings.xml**: Sirve para colocar los textos de la App.
- **Styles.xml**: Sirve para poner estilos, como un CSS.

Vamos a ir a **activity_main.xml**. Aquí es donde se ‘monta’ la ventana con su **Button** y su **TextView**. Podemos hacerlo de dos maneras: con la herramienta visual que nos da Android Studio, o **a mano** editando el fichero **activity_main.xml**. Lo más sencillo es hacerlo con la herramienta visual, pero hay que saber modificar el xml por lo que pudiera ocurrir.

La herramienta visual se ve así. Fíjate que por defecto te añaden un **ConstraintLayout** y un **TextView** con el texto “Hello World”.



El código se ve así. Si te fijas, es lo mismo: un **ConstraintLayout** y un **TextView** con el texto “Hello World”.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/a
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

Utilizando la herramienta visual, añade un **Button** a la ventana. El **Button** tiene que quedarte dentro del **ConstraintLayout**, pero no importa dónde lo coloques por el momento.

Vamos ahora al **strings.xml**, donde añadiremos los textos que queremos usar en la App. Si te fijas, el name funciona como si fuese una ID. Los textos que queremos añadir son:

```
<resources>
    <string name="app_name">ButtonAndTextView</string>
    <string name="text_ok">OK</string>
    <string name="text_pulsa">Pulsa el boton</string>
    <string name="text_hola_mundo">Hola Mundo</string>
</resources>
```

Volvemos, a la herramienta visual. En la barra de la derecha están los Atributos. Esos Atributos son en realidad los del código xml, pero puestos así para que se trabaje con ellos de forma más sencilla. Deberíamos de rellenar lo siguiente:

- Para el **Button**:
 - ID: button
 - Text: @string/text_ok

- Para el **TextView**:
 - ID: textView
 - Text: @string/text_pulsa

Si te fijas, los textos que aparecen ahora en la herramienta visual los sacamos del **strings.xml**. Ésta es la forma que tiene Android de tener varios idiomas en una misma App, por ejemplo.

Si miras ahora el código, verás que el **activity_main.xml** ha cambiado. Si recuerdas lo básico de xml, te darás cuenta de que puedes leer y modificar directamente la ventana desde aquí.

Por el momento, ya hemos acabado con la **parte visual** de la App.

Tercer paso – La lógica de negocio

Vamos ahora a la clase **MainActivity.java**. En esta clase se describe el comportamiento cada uno de los elementos de la parte del diseño. Por defecto, el Android Studio nos da este código:

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate( savedInstanceState );  
        setContentView( R.layout.activity_main );  
    }  
}
```

Este código no debe de ser borrado. La función **onCreate** se lanza siempre ella sola cuando se inicia la App. No es un constructor de una clase normal de Java, pero se usa como si lo fuera. Las dos líneas que tiene siempre tienen que dejarse como están al principio del método, nosotros añadiremos nuestro código debajo.

El texto **R.layout.activity_main**, es una referencia al fichero **activity_main.xml**, nuestra ventana. Ésta es la manera que tiene Android de saber qué Activity va con qué xml. Si te fijas, el texto se corresponde con la ruta **app/res/layout/activity_main.xml**.

Para poder utilizar el **Button** y el **TextView**, es necesario que Java los referencie. Para ello, Creamos una variable **Button** y una **TextView**, y buscamos los widgets por ID:

```
public class MainActivity extends AppCompatActivity {  
  
    private Button myButton = null;  
    private TextView myTextView = null;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate( savedInstanceState );  
        setContentView( R.layout.activity_main );  
  
        myButton = (Button) findViewById( R.id.button );  
        myTextView = (TextView) findViewById( R.id.textView );  
    }  
}
```

De nuevo, **R.id.button** es una referencia a un recurso cuya ID es *button*.

No nos queda más que indicar el evento al **Button** que hará cambiar el texto del **TextView**. Para ello, creamos un método `onClick ()` y escribimos el código.

```
public void onClick (View v){
    // el texto actual del TextView
    String textViewText = myTextView.getText().toString();
    // Texto 'Pulsa el boton' del strings.xml
    String textPulsa = getString(R.string.text_pulsa);
    // Texto 'Hola Mundo' del strings.xml
    String textHola = getString(R.string.text_hola_mundo);
    // Comparamos y cambiamos
    if (textViewText.equalsIgnoreCase(textPulsa)){
        myTextView.setText(textHola);
    } else {
        myTextView.setText(textPulsa);
    }
}
```

De nuevo, **R.string.text_pulsa** es la forma que tenemos de sacar los textos del fichero de Recursos **string.xml**.


Este método `onClick ()` es en realidad un evento, pero nunca se va a ejecutar si no se lo asignamos al botón. Para ello, primero tenemos que decirle a la clase **MainActivity** que tiene que implementar la Interfaz **View.OnClickListener**.

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
```

Y luego, añadir el evento al **Button** (los click del botón van al método `onClick ()` ellos solos)

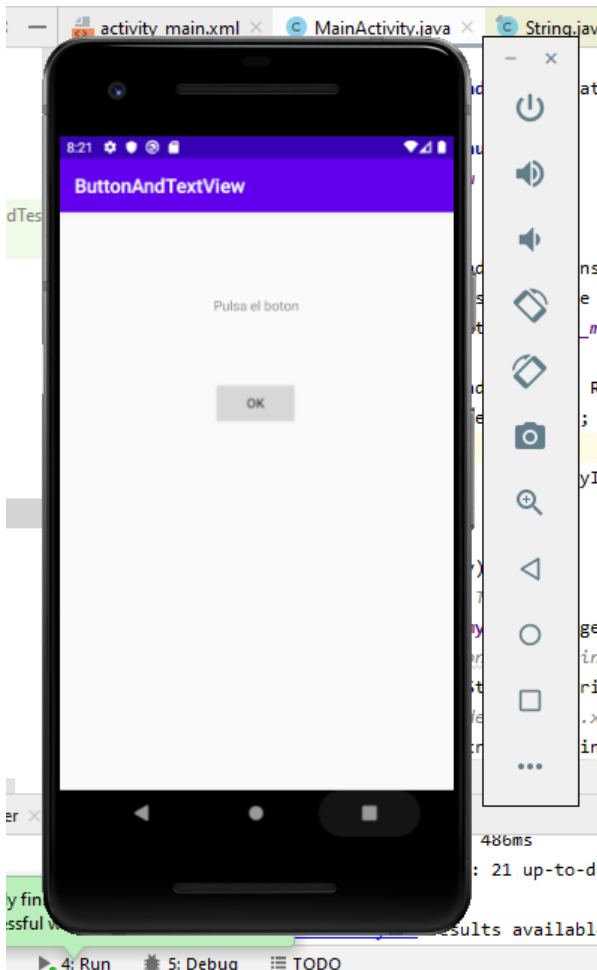
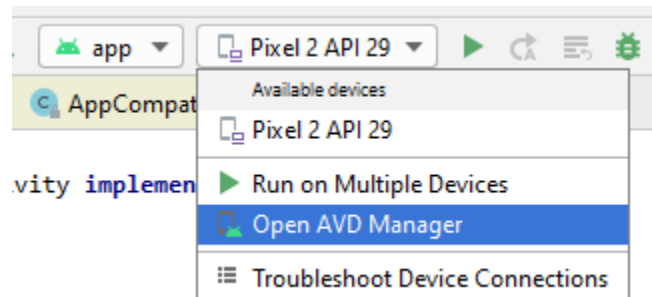
```
myButton = (Button) findViewById( R.id.button );
myButton.setOnClickListener(this);
```

Existen varias formas de añadir eventos a cada uno de los elementos que vayas a utilizar. Éste es el más sencillo, pero en realidad no se hace así (manualmente) porque las cosas tienden a complicarse bastante cuando tienes *actividades* con siete eventos diferentes, clases anónimas, etc. Por el momento, nos vale.

Hemos terminado. Sólo nos queda darle al **Make Project**  y esperar a que se construya la App. Es un proceso que puede llevar bastante tiempo, y es un paso previo a instalar la App en un dispositivo (o emularla).

Cuarto paso – Emulación

Vamos a **emular** un dispositivo Android. Si no tienes ningún dispositivo ya definido en tu Android Studio, puedes bajarte uno con la opción **Open AVD Manager**. En mis ejemplos yo tengo por defecto el **Pixel 2 API 29**, pero puedes tener tantos como quieras. De hecho, tienes que asegurarte de que App funciona como se supone en todos los dispositivos del mercado...



Una vez bajado y seleccionado, bastaría con darle al botón de Run ► para que se ejecutase el emulador.

La **emulación** es un proceso muy costoso en recursos para un equipo. Va lento, puede colgarse si no tienes memoria, un disco duro rápido, un procesador bueno, o si empiezas a darle a los botones. La verdad es que es desesperante tener que andar apagando para volver a hacer pruebas. Nunca te fíes de volver a darle al Run tras un cambio en el proyecto.

Afortunadamente, el Android Studio incorpora una herramienta de Debug que funciona muy bien.

Quinto paso – Instalando la app en tu móvil (Opcional)

Si deseas instalar la app en tu dispositivo Android personal, tendrás que tener un cable USB y esperar a que el Android Studio te lo detecte. Debería de ser algo automático. Será necesario que antes **habilites tu dispositivo en modo desarrollador**.

Cada modelo lo hace de una manera diferente, deberías de mirarlo en Internet. En mi caso, tengo que acceder a **Ajustes → Sistema → Acerca del teléfono**, y bajar hasta la opción de **Número de compilación**. Una vez allí, **pulso siete veces seguidas** sobre esta opción. Si ya lo has hecho una vez, no hay que volver a hacerlo más adelante.

A continuación, tengo que conectar el cable y habilitar la depuración: **Ajustes → Sistema → Opciones del Desarrollador → Habilitar Depuración USB**. Si todo ha ido bien, el Android Studio detectará mi móvil y me aparecerá en la pestaña del **emulador**. Si tienes algún problema, el **Open AVD Manager** me permite hacer que el Android Studio lo busque y me informa de cualquier problema al respecto.

Ten en cuenta que dado que vas a programas, desplegar y debugear sobre tu móvil; veces se queda un montón de ‘basura’ en memoria. Se suele solucionar desinstalando a mano la app. Es recomendable que lo hagas con frecuencia.

No es obligatorio que uses tu móvil en mis clases, pero si lo haces, es cosa tuya. Pueden ocurrir cosas imprevistas, por ejemplo, si dejas el móvil en modo desarrollador todo el rato, responderá muy lento y se agotará la batería. Estás avisado.