

# 1. Tu primera aplicación en Android

Este primer capítulo del curso está lleno de nuevos conceptos, así que no te preocupes si no los entiendes todos al final de este capítulo ya que se volverán a tratar con más detalle en el transcurso del curso.

La primera aplicación que vas a crear se llama GeoQuiz. GeoQuiz es una aplicación muy sencilla, hecha para poner a prueba los conocimientos de geografía del usuario. El usuario deberá seleccionar TRUE o FALSE a la pregunta que le aparezca por pantalla y GeoQuiz le informará sobre si ha contestado correctamente o por el contrario ha errado.



## Conceptos básicos de la aplicación

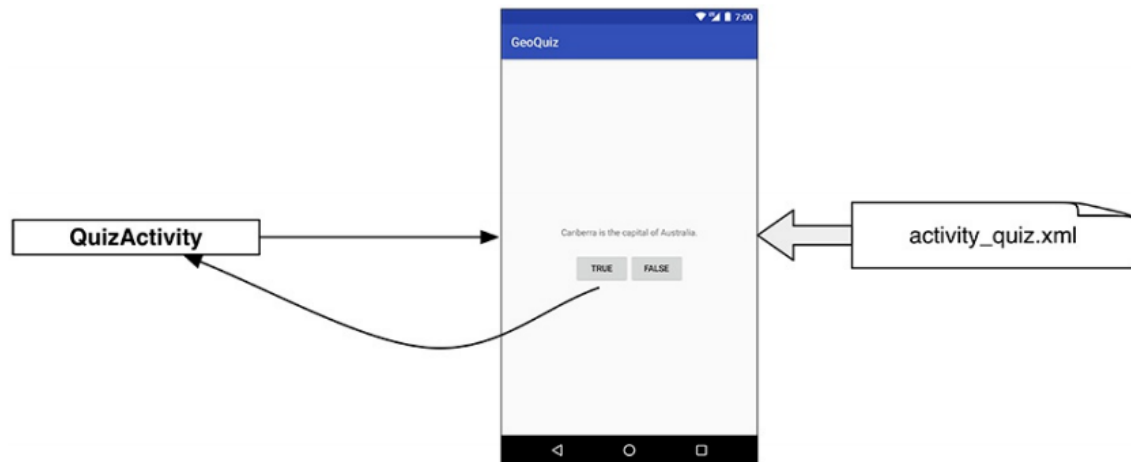
La aplicación GeoQuiz consistirá en una *activity* y un *layout*.

- Una *activity* es una instancia de **Activity**, una clase del Android SDK. La *activity* es la responsable de controlar la interacción del usuario con la pantalla y su información.

Tu escribirás sub clases de **Activity** para implementar funcionalidades que requiera tu aplicación, como es el caso de GeoQuiz, en la que sólo necesitarás implementar una única sub clase de **Activity** a la que llamaremos **QuizActivity**. Ésta se encargará de controlar la interficie gráfica (también llamada *UI*) vista anteriormente.

- Un *layout* define un conjunto de elementos gráficos y su posicionamiento en la pantalla. Este layout se construirá mediante un archivo *XML*. Como GeoQuiz tan sólo tiene una pantalla, únicamente deberemos crear un fichero XML llamado **activity\_quiz.xml**, que tendrá definidos todos los botones u otros elementos que se requieran para formar la pantalla que hemos visto antes.

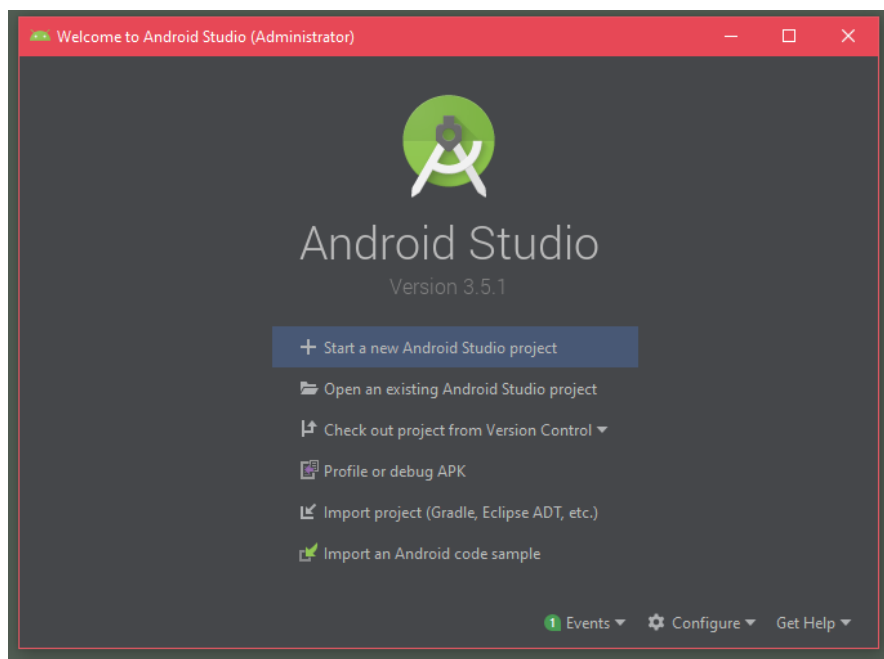
La relación entre la activity **QuizActivity** y **activity\_quiz.xml** será la siguiente:



Teniendo esto en mente, vamos a construir nuestra primera aplicación, empezando por crear el proyecto Android.

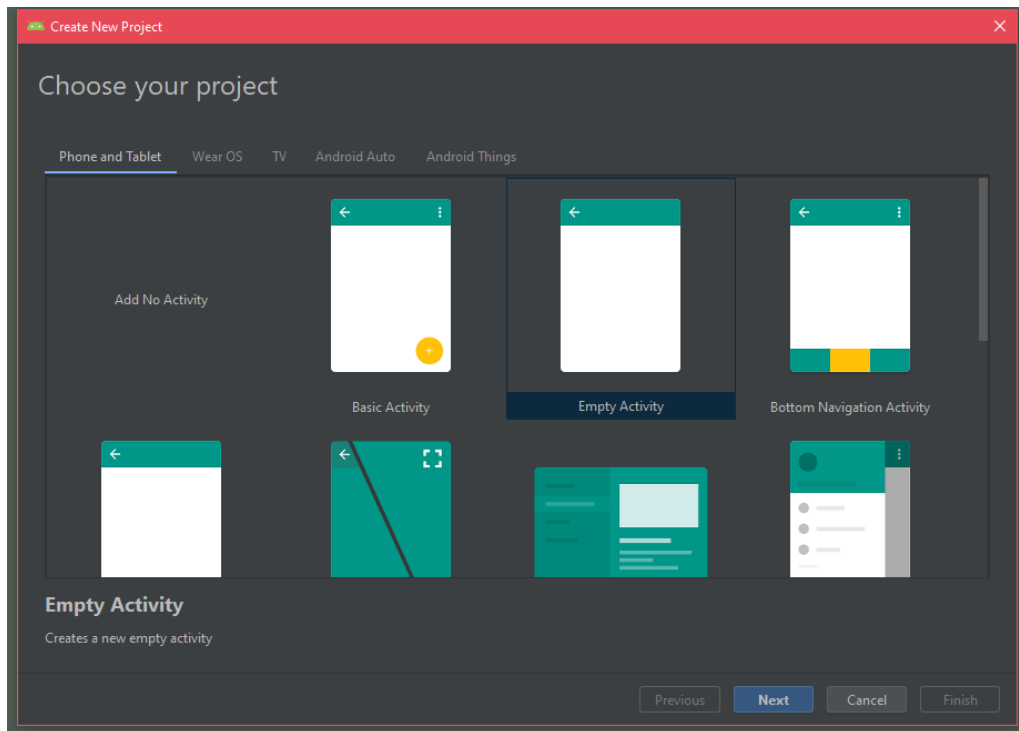
## Creando un proyecto Android

Un proyecto Android contiene los archivos necesarios para crear una aplicación. Para crear el proyecto deberemos abrir **Android Studio**.

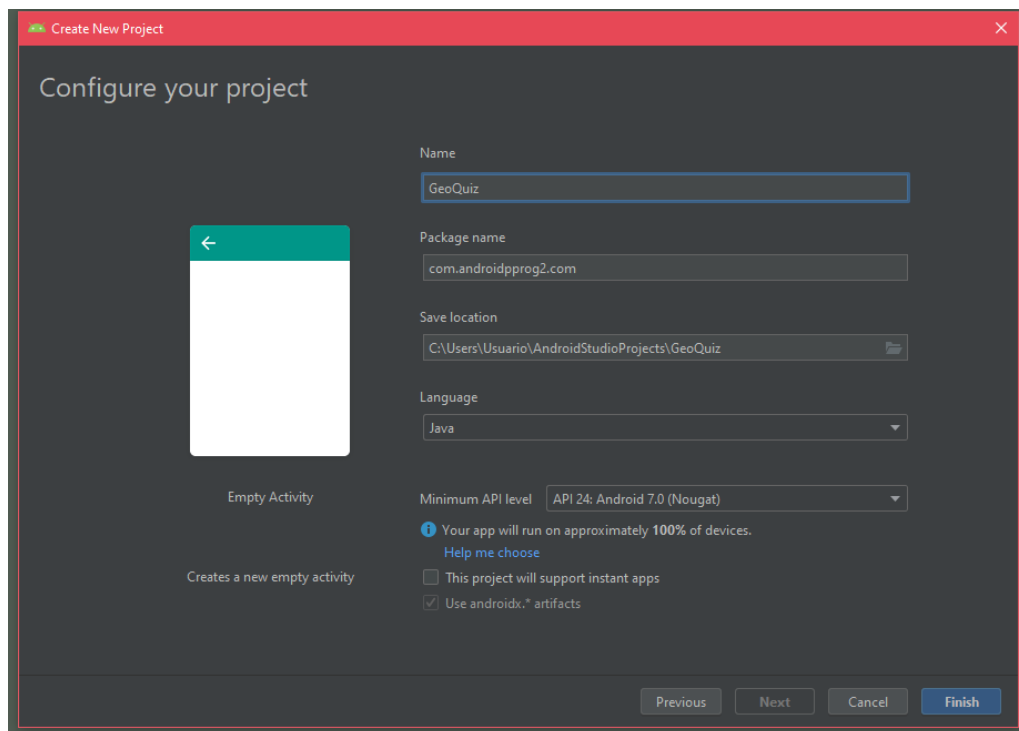


Nota: Cuando el software esté listo para una nueva **actualización**, nos aparecerá la pestaña **Events** en la parte inferior del menú de inicio de Android Studio.

Para crear el proyecto deberemos clicar en “*Start a new Android Studio project*” y nos aparecerá la siguiente pantalla.



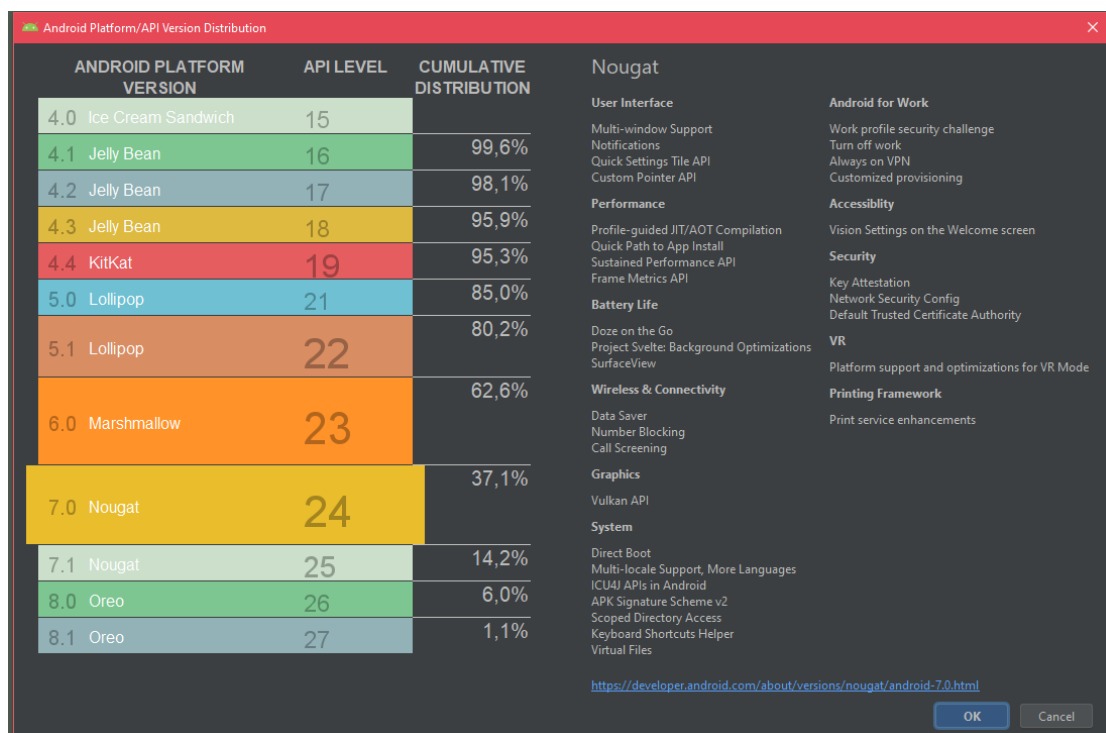
En esta pantalla **deberemos elegir el formato de la primera actividad, que se lanzará al arrancar la aplicación**. Como estamos iniciándonos en el desarrollo de aplicaciones Android, siempre inicializaremos la aplicación con una actividad vacía. Más adelante, cuando ya seamos unos expertos en este tipo de desarrollo, podremos iniciar la aplicación con la actividad que más nos convenga, pero ahora vamos a aprender a programarlo todo desde cero.



Una vez elegida la primera pantalla (activity) de la aplicación deberemos configurar el proyecto.

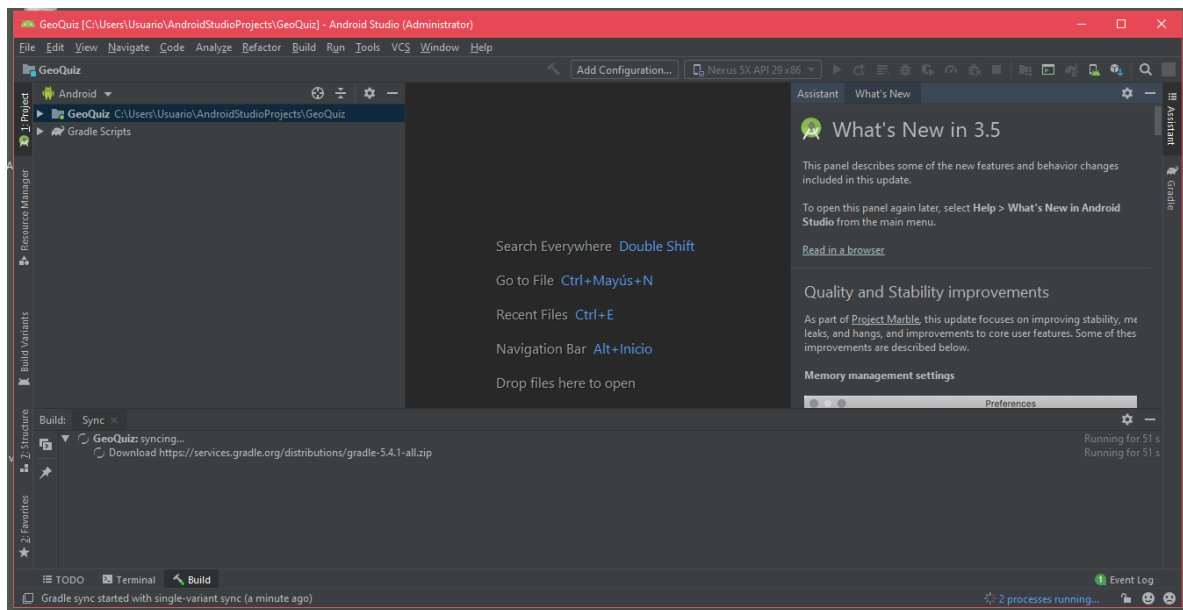
Los aspectos a configurar serán los siguientes:

- **Name:** GeoQuiz, el nombre de la aplicación.
- **Package Name:** com.androidpprog2.com, será el nombre que le asignemos al paquete. A la hora de desarrollar un proyecto real como empresa, será importante configurar este campo como el dominio de la empresa con algunos identificadores del proyecto. Este campo mantiene los nombres de los paquetes únicos y distingue las aplicaciones entre sí en un dispositivo y en Google Play.
- **Save location:** Será el directorio destino en el que crearemos el proyecto Android.
- **Language:** Java. En la asignatura de “Proyectos de Programación II” aprenderemos a programar una app Android mediante Java, no Kotlin. Ambas son igual de aceptadas en entornos de desarrollo.
- **Minimum API Level:** API 24 (Android 7.0 – Nougat). En este campo especificaremos la versión de Android mínima del dispositivo para poder ejecutar nuestra aplicación. Es importante saber que si se lanza una nueva aplicación con una de las últimas versiones de Android, no todos los dispositivos serán capaces de ejecutarla ya que tales dispositivos pueden permanecer obsoletos (en cuanto a actualizaciones del sistema operativo Android) y por ello deberemos buscar un balance entre la versión mínima de Android y el número de dispositivos que serán capaces de correr la aplicación. Si tenemos dudas al respecto podremos clicar en “help me choose” y nos mostrará un gráfico de uso de cada versión de Android, y además nos mostrará información al respecto de cada versión.



- **This project will support instant apps:** No llegaremos a utilizar esta opción ya que se utiliza para poder modular el proyecto de tal forma que cada usuario pueda correr la aplicación sin la necesidad de descargar la app completa, tan sólo los módulos que vaya a utilizar, mediante la redirección dentro de la app con links a los distintos módulos en un server.

Clicamos en “Finish” para crear el proyecto y esperamos a que este sea creado:



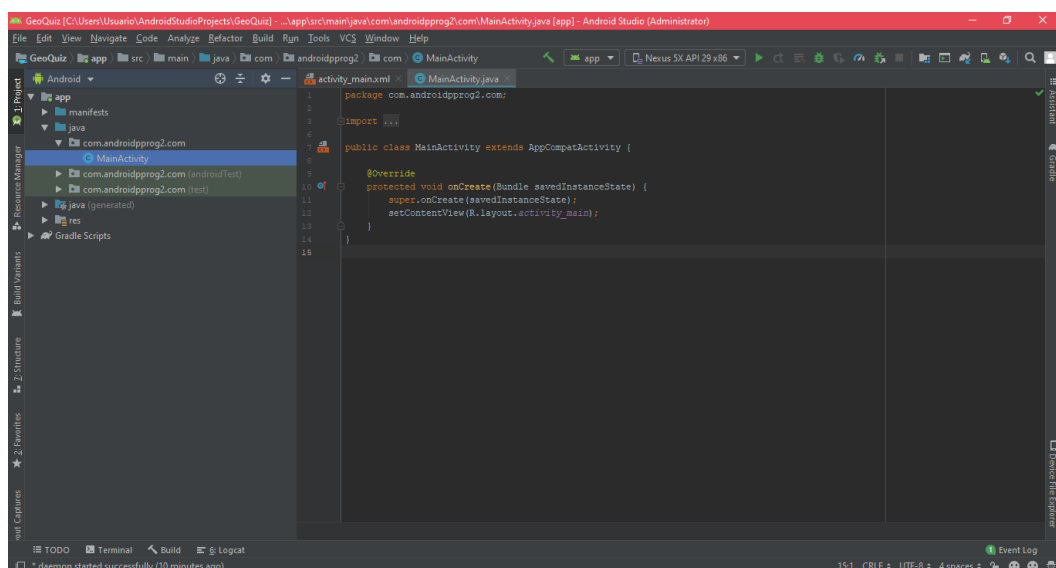
Mientras se cree el proyecto, al ser la primera vez, se descargarán todos los paquetes y dependencias necesarias. Además nos mostrará un panel en la parte derecha de la pantalla, titulado “What’s New” para informarnos de las novedades de la última actualización de Android Studio.

## Navegando por Android Studio

A continuación, se describirán las distintas partes de Android Studio.

En la parte izquierda de la pantalla podemos ver un panel, que podremos esconder, en el que tendremos la distribución de los distintos archivos del proyecto. Este panel es llamado **project tool window**. Por defecto, Android Studio nos ha creado un archivo llamado **MainActivity**, que contiene el código de inicialización de la actividad (en este caso MainActivity será la activity que se mostrará al ejecutar la app).

La parte principal del programa es el editor, en el que podremos modificar el propio código de la aplicación.



## Preparando la UI de la aplicación

Abrimos el archivo situado en *app/res/layout/activity\_main.xml*.

Este archivo representa una interfaz gráfica, que asociaremos a una activity, en nuestro caso será la interfaz gráfica de la MainActivity.

```
<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>
```

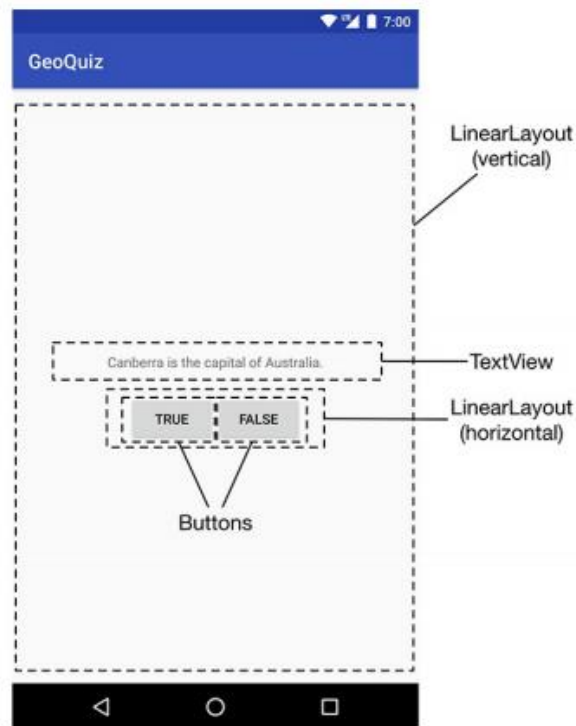
La interfaz gráfica por defecto está formada por dos **widgets**: Un **ConstraintLayout** y un **TextView**.

Los widgets son los bloques que compondrán la interfaz gráfica, y cada widget puede mostrar texto, mostrar gráficos, interactuar con el usuario o albergar otros widgets según una disposición determinada. Todo widget es una instancia de la clase **View** o de alguna de sus subclases.

El Android SDK incluye muchos widgets distintos que se pueden configurar de forma que nuestra aplicación resulte vistosa.

La interfaz creada por defecto, como es de esperar, no es la interfaz que deseamos para nuestra aplicación. La interfaz de la MainActivity deberá contener como mínimo los siguientes widgets:

- Un **LinearLayout** en orientación vertical.
- Un **TextView**.
- Un **LinearLayout** con orientación horizontal.
- Dos **Buttons**.



Ahora debemos definir tales widgets en nuestro archivo **activity\_main.xml**.

Cuando nos dirijamos al archivo `app/res/layout/activity_main.xml`, podremos visualizar dos pantallas distintas.

La primera de estas pantallas que podemos ver es la de **text**, en la que podremos modificar, añadir o eliminar widgets mediante código (en XML).

La segunda pantalla es la de **design**, en la que podremos modificar, añadir o eliminar widgets mediante un editor gráfico. Verdaderamente, es una herramienta muy útil para empezar a programar interfaces gráficas en Android.

En la pantalla de text procedemos a eliminar todo su contenido, y en su lugar copiamos el código siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
```

```

        android:id="@+id/true_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/true_button"/>

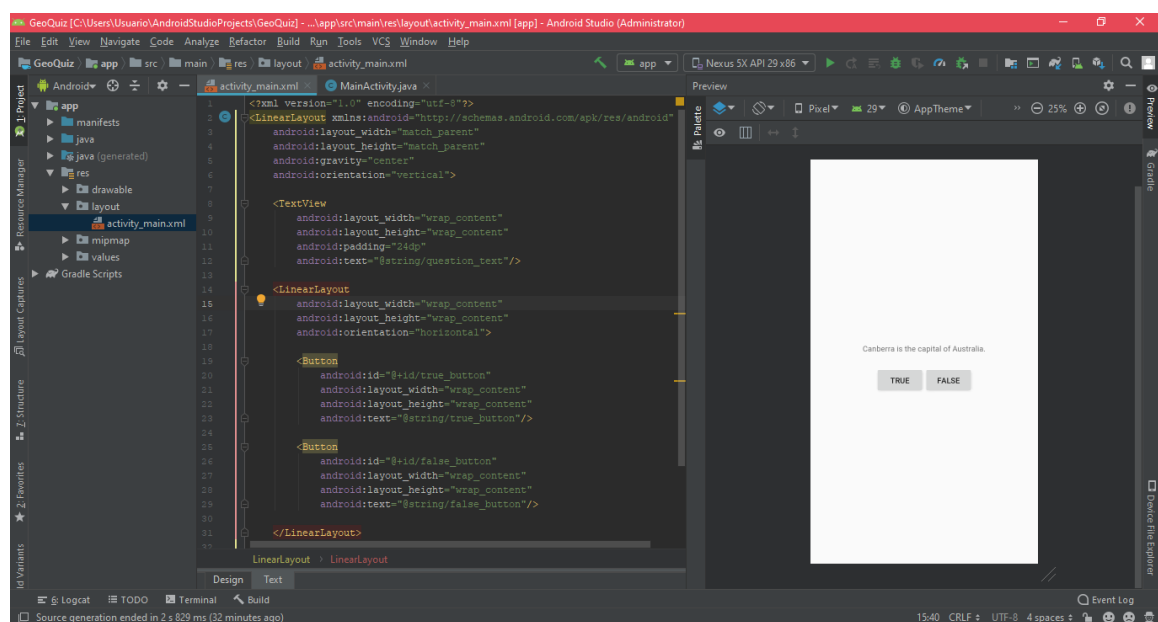
<Button
    android:id="@+id/false_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/false_button"/>

</LinearLayout>
</LinearLayout>

```

No es necesario comprender el código que acabamos de copiar, lo explicaremos a continuación. En todo caso, que no cunda el pánico cuando aparezcan tres errores en las líneas en las que pone *android:text* ya que será solucionado más adelante.

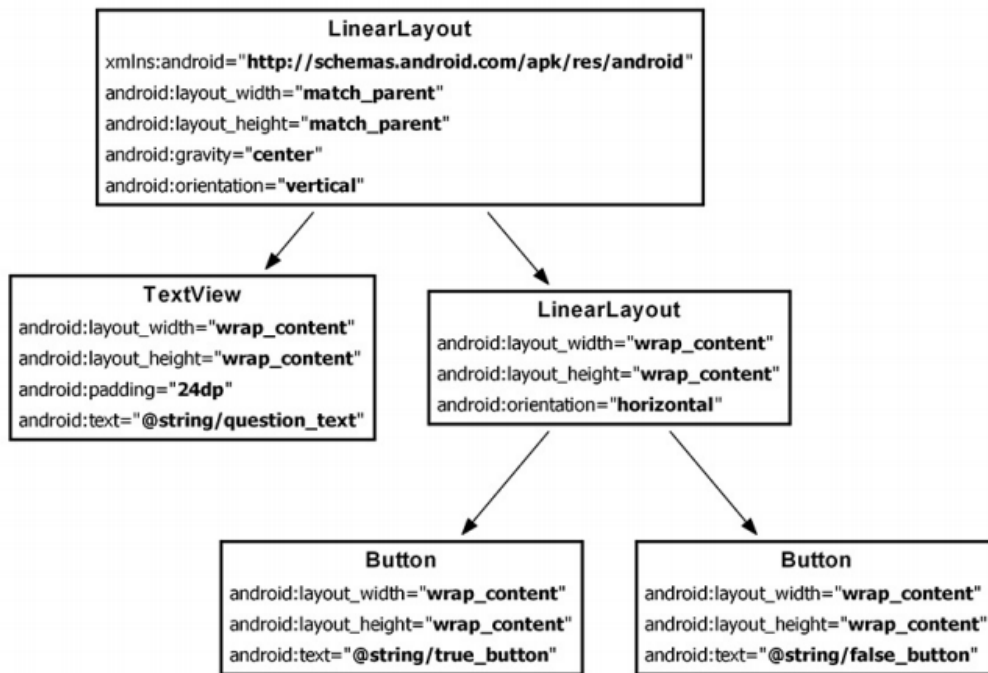
Visualicemos ahora el diseño que acabamos de programar con la herramienta de previsualización de Android Studio. Encontraremos esta herramienta en la barra lateral derecha de la pantalla de *text* del archivo XML.



Podemos apreciar que los elementos ya están bien situados en la pantalla, pero carecen de texto (ignora que en la imagen si que aparecen los textos de cada widget).

Los atributos de cada widget nos permitirán darles formato a los elementos, tanto gráficamente como en funcionalidades.





El elemento padre de toda la vista es un LinearLayout. El widget LinearLayout hereda de una subclase de View llamada **ViewGroup**, que se caracteriza por contener y situar distintos widgets en su interior. Los distintos ViewGroup que veremos a lo largo de la asignatura son los siguientes:

- **LinearLayout**: Para organizar los elementos uno tras otro en línea (tanto horizontal como en vertical).
- **FrameLayout**:
- **TableLayout**
- **RelativeLayout**
- **ConstraintLayout**

Cuando un ViewGroup contiene otros widgets en su interior, a éstos se les referencia como hijos de este ViewGroup. En nuestro caso, el TextView y el otro LinearLayout son hijos de este primer LinearLayout. Finalmente vemos que el segundo LinearLayout tiene dos hijos que son Button.

## Atributos de los widgets

Dos de los atributos más importantes de cada widget son su altura y su anchura, ya que estos deberán ocupar una región de la pantalla.

- *android:layout\_width*: Representa la anchura del widget.
- *android:layout\_height*: Representa la altura del widget

Ambos atributos podrán ser asignados con 2 tipos distintos de valores:

- **match\_parent**: Se expandirá hasta encontrar a su widget contenedor.

- `wrap_content`: Se expandirá tanto como necesite su contenido para caber en él.

Además, cada widget podrá tener otros muchos atributos, entre ellos: *margin*, *padding*, *orientation*, *text*, *backgroundColor*... Es importante que el alumno investigue y juegue con los distintos elementos básicos y sus atributos para familiarizarse con la plataforma.

## Creación de *string resources*

Es lógico pensar que cuando una empresa desarrolla una aplicación la hace para un idioma y posteriormente ésta se traduce sola a los distintos idiomas a los que se quiera lanzar la aplicación.

La aplicación deberá saber en qué idioma debe mostrarse dependiendo del idioma en el que esté configurado el teléfono, accediendo a un fichero llamado *strings.xml*. Este fichero se encuentra siempre en el directorio *res/values/strings.xml* del proyecto de Android Studio.

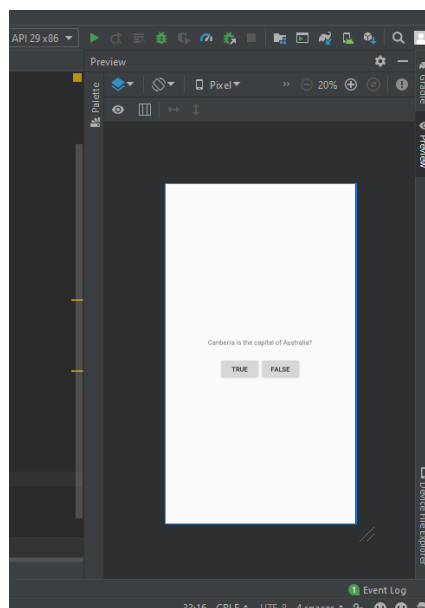
En él podremos definir strings en un idioma, para más tarde hacer las traducciones a otros idiomas, de la siguiente manera:

```
<resources>
  <string name="app_name">GeoQuiz</string>
  <string name="question_text">Canberra is the capital of
Australia.</string>
  <string name="true_button">True</string>
  <string name="false_button">False</string>
  <string name="correct_toast">Correct!</string>
  <string name="incorrect_toast">Incorrect!</string>
</resources>
```

Dependiendo de la versión de Android Studio es posible que tengas alguna string adicional al inicio: No la borres ya que puede provocar algún fallo en cascada en otros archivos.

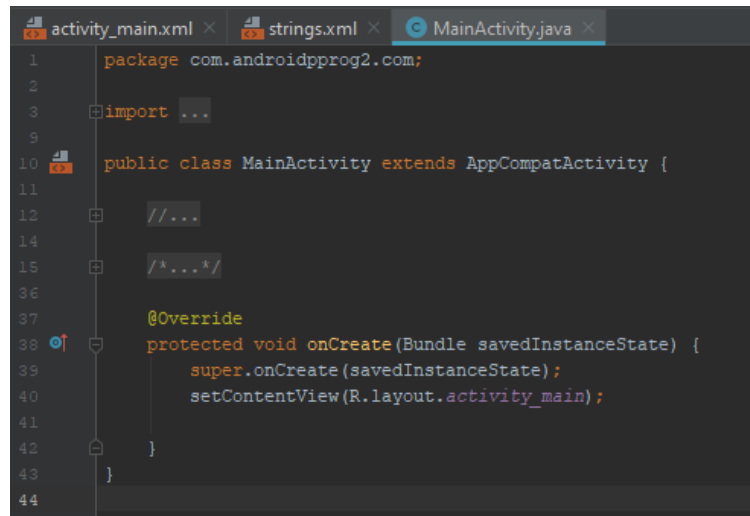
## Previsualización de la pantalla

Para poder visualizar cómo estamos diseñando la pantalla sin tener que testear la aplicación entera, Android Studio nos ofrece una herramienta en la pantalla de edición del archivo XML, que podremos encontrar en la parte derecha de la pantalla.



## Del Layout XML a View Objects

¿Cómo es posible que los elementos definidos en el archivo *activity\_quiz.xml* puedan verse en la aplicación? La respuesta está en la clase **MainActivity**, que se encuentra en el directorio *app/java* de nuestro proyecto.



```
1 package com.androidpprog2.com;
2
3 import ...
4
5
6
7
8
9
10 public class MainActivity extends AppCompatActivity {
11
12     //...
13
14     /*...*/
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37 @Override
38 protected void onCreate(Bundle savedInstanceState) {
39     super.onCreate(savedInstanceState);
40     setContentView(R.layout.activity_main);
41 }
42
43 }
44
```

Podemos ver que la clase **MainActivity** contiene por defecto un método llamado **onCreate(Bundle)**. Este método es llamado cuando una instancia de la clase **Activity** es creada.

Para poder definir cual es la view de nuestra screen, será necesario llamar al método **setContentView(int layoutResId)**, el cual buscará en la carpeta de *res/* algún elemento que tenga el ID que queramos.

Este método infla la layout y la pone en pantalla.

NOTA: Cualquier elemento de la aplicación que no sea código será un resource (imágenes, audios y archivos XML). También lo serán aquellas strings que hayamos definido.

## Identificar Widgets

Con tal de identificar widgets para así poder detectar cualquier interacción con ellos o bien cambiar su estado, será necesario que les asignemos un id.

Cómo se ha visto en alguna captura anterior, le hemos definido un atributo a los botones en el archivo XML de la *activity\_quiz.xml*:

- `android:id="@+id/true_button"`
- `android:id="@+id/false_button"`

El símbolo + después del @ significa que estamos creando un ID.

Ahora, en la clase **MainActivity**, queremos recuperar esos botones para así detectar cuando han clicado alguno de ellos.

```

public class MainActivity extends AppCompatActivity {

    /*...*/

    private Button mTrueButton;
    private Button mFalseButton;

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

```

Si aparece algún error en la definición de ambos botones, será porque no hemos importado la clase Button a nuestra MainActivity. Debemos escribir la siguiente línea para poder importarla.

```
import android.widget.Button;
```

Cabe la alternativa de apretar *alt + Enter* para que el error se arregle solo e importe la clase Button a la MainActivity.

## Referenciar los Widgets

Hemos creado dos botones en la parte gráfica (xml), y hemos creado dos botones en la parte java, pero, ¿cómo los conectamos entre sí para que sean el mismo objeto?

Muy fácil, lo que tendremos que hacer será referenciarlos con el id que les hemos asignado.

```

public class MainActivity extends AppCompatActivity {

    /*...*/

    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}

```

Ahora sólo nos faltará capturar el momento en el que el usuario apriete alguno de los dos botones.

## Configurando Listeners

Para capturar cuando el usuario clique un botón (empecemos con el botón de TRUE), tendremos que asignarle un **OnClickListener** al propio botón.

```

public class MainActivity extends AppCompatActivity {

    /*...*/

    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Ha clicado en el botón de TRUE
            }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}

```

Hemos asignado un `OnClickListener` al botón de TRUE, y éste captura cuando ha sido apretado el botón, ejecutando el código situado dentro de su propio método `onClick(View)`.

Haremos exactamente lo mismo con el botón de FALSE, así ambos ya podrán detectar cuando han sido clicados.

## Toasts

Cuando el usuario apriete alguna de las dos opciones, se le deberá notificar si ha acertado a la pregunta o no. Para tal propósito le mostraremos un Toast.

Un **Toast** no es más que un mensaje que aparece de forma emergente desde la parte inferior de la pantalla e informa brevemente al usuario de alguna acción que éste ha realizado.

El mensaje que el Toast muestre también deberá estar albergado en el fichero `strings.xml`, como TODAS las strings que se muestren por pantalla. **Nada deberá estar HARDCODED.**

Para mostrar el Toast, deberemos escribir el siguiente código en el método `onClick` del listener.

```

mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText( context: MainActivity.this,
                        R.string.correct_toast,
                        Toast.LENGTH_SHORT).show();
    }
});

```

Lo haremos de la misma forma para el botón de FALSE, y nos resultará en algo así.

```

public class MainActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(context MainActivity.this,
                               R.string.correct_toast,
                               Toast.LENGTH_SHORT).show();
            }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(context MainActivity.this,
                               R.string.incorrect_toast,
                               Toast.LENGTH_SHORT).show();
            }
        });
    }
}

```

A la función `makeText()`, le pasaremos el contexto en el que se deberá mostrar el Toast (es decir, nuestra pantalla relativa la `MainActivity`), el ID de la string que queremos que se muestre, y el tiempo que se deberá mostrar. Finalmente la función `show()` mostrará el Toast.

# Challenges

Si el alumno se ve con ganas puede descubrir por si mismo los siguientes puntos:

- Tener varias preguntas y cambiar la pregunta en cuanto sea respondida.
- Personalizar el Toast para que éste aparezca en la parte superior de la pantalla en vez de abajo.
- Asignar un valor de respuesta correcta a cada pregunta y mostrar el Toast en función de si ha acertado o no a la pregunta.