índice

Enunciado Manual Android Java

Manual_AndroidJava

Empezar tarea

Fecha de entrega Miércoles a las 23:59 **Puntos** 0
Entregando una carga de archivo **Tipos de archivo** pdf
Disponible hasta el 27 de nov en 23:59

Desarrolla un manual sobre el uso de Android Studio y su programación en Java. El manual debe contener explicaciones, capturas de pantalla o fragmentos de código, de forma que quede explicado cómo se hace todo lo que se ha visto en ejercicios prácticos en clase.

Un índice que puede servirte como guía puede ser el siguiente (aunque puedes adaptarlo a tus necesidades):

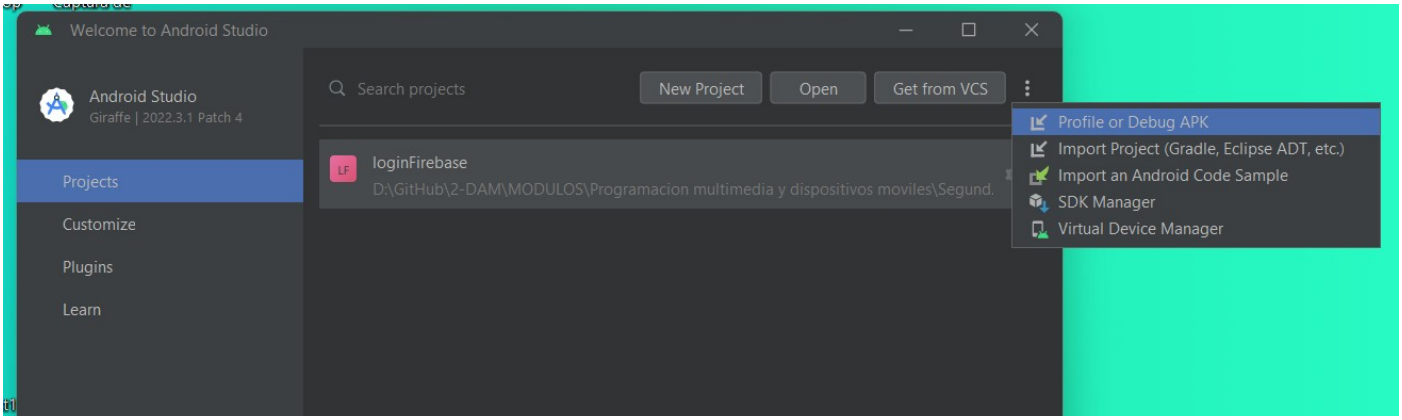
- Estructura y conceptos básicos:
  - Estructura de un proyecto de Android.
  - Ficheros importantes en un proyecto Android con Gradle.
  - Instalación de un emulador o dispositivo de pruebas.
  - Concepto de activity, ficheros que contiene y cómo crear una nueva.
- XML:
  - Interfaz en XML.
  - Otros XML importantes.
- Java:
  - Eventos de botón.
  - La clase Intent:
    - Abrir otra activity
    - Abrir correo electrónico
    - Abrir teléfono
    - Abrir navegador
    - Paso de datos (intent.putextra).
- SQLite:
  - Crear base de datos y tabla desde una clase de Java.
  - Añadir y leer registros.
  - Cómo ver la tabla creada (donde está el fichero y cómo abrirlo).
- Backend*.

*Se irá ampliando este índice conforme vayamos avanzando en el temario.

Estructura y conceptos básicos

Descargar Android Studio

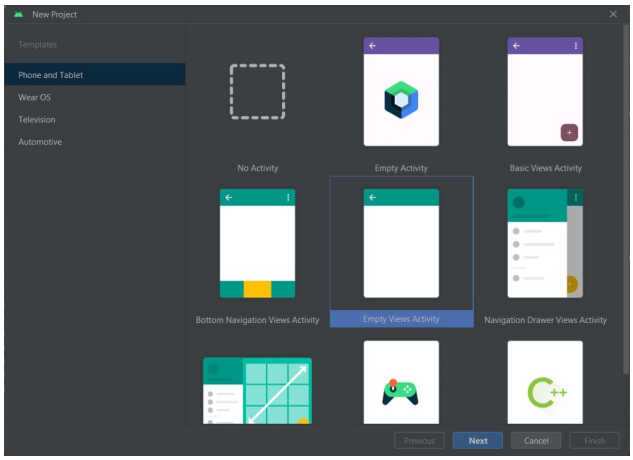
Opciones de Android Studio



- **New Project** crear un nuevo proyecto
- **Open** abrir un proyecto existente
- **Get from VCS** clonar un proyecto desde un sistema de control de versiones (como por ejemplo github)
- **Profile or Debug APK** analizar o depurar un archivo APK
- **Import Project** importar un proyecto existente
- **Import an Android Code Sample** importar código de muestra de Android
- **SDK Device Manager** permite crear y gestionar dispositivos virtuales para pruebas

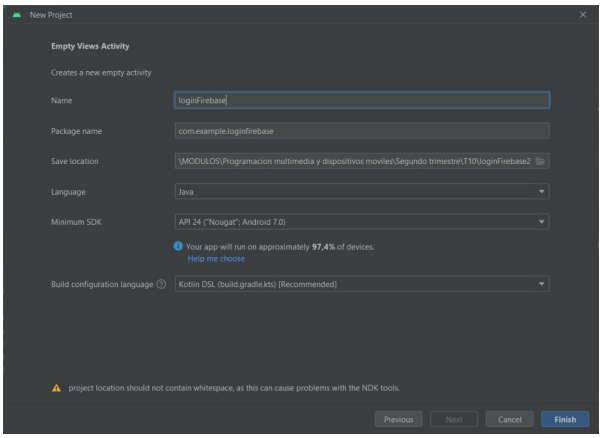
Crear un proyecto:

Creamos el proyecto con el tema **Empty Views Activity.** Empieza con la Activity en blanco.



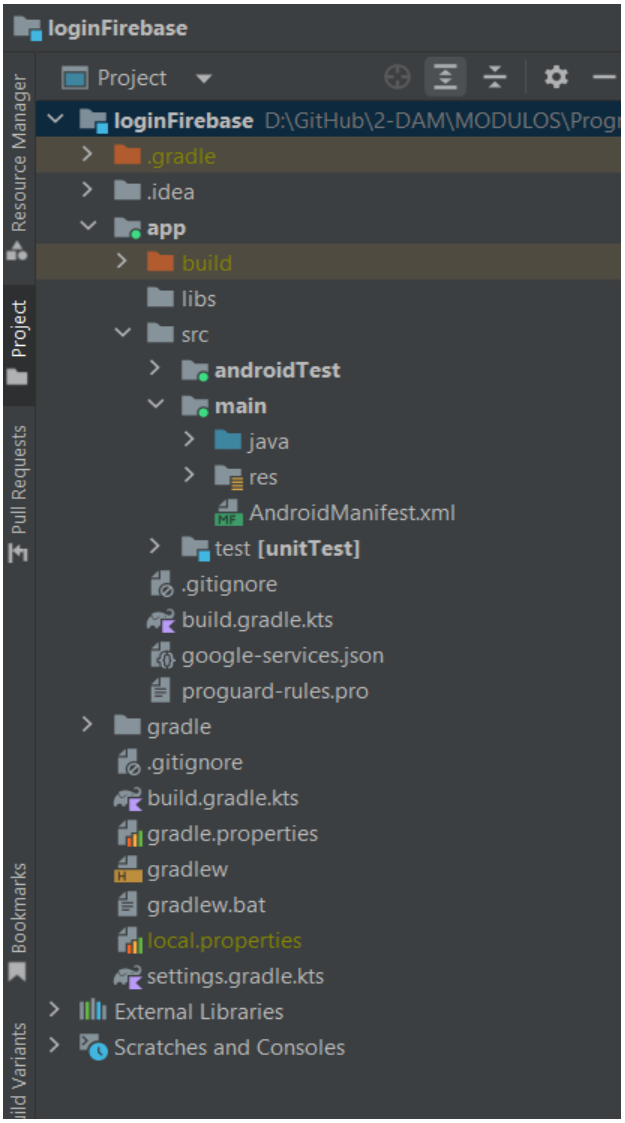
Le ponemos un nombre a nuestro proyecto preferiblemente con "_ "o todo junto separando las palabras con mayúsculas.

Tener en cuenta: que deberemos poner el nombre del proyecto y en la ruta ponerlo también para que el proyecto se cree dentro de una carpeta. Si no se pondrán todos los directorios del proyecto dispersos.



Estructura de un proyecto de Android.

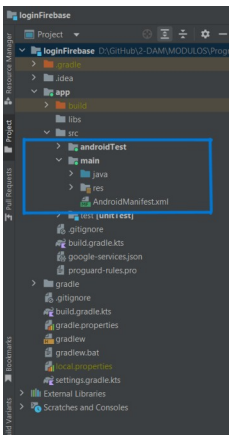
Perspectiva Project



Un proyecto Gradle en Android Studio (Java).

Contiene los directorios que se pueden ver en la imagen.

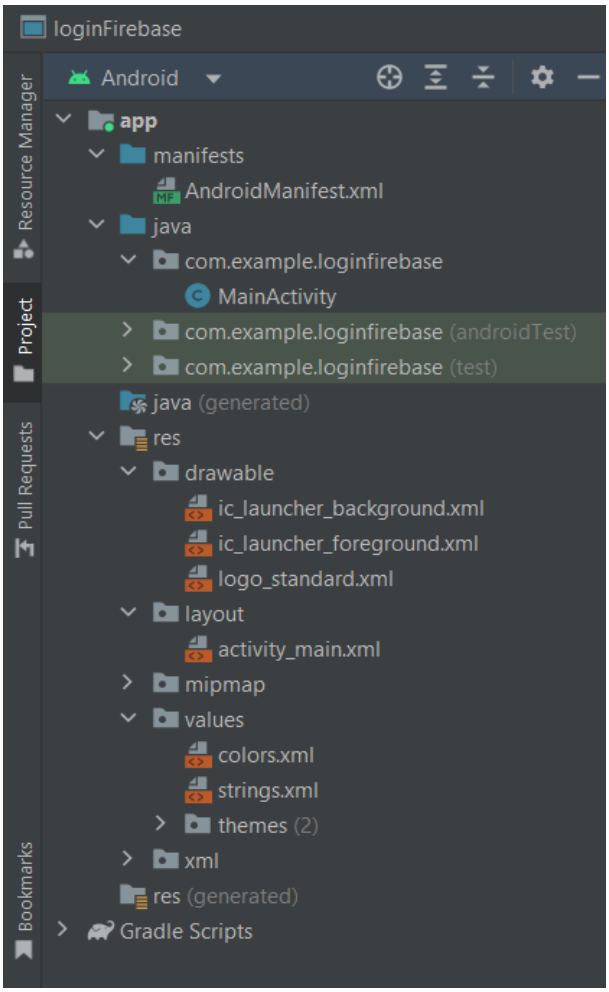
- **build.gradle** Este archivo se encarga de configurar cómo se compila y empaqueta nuestro proyecto, así como de gestionar tareas y dependencias.
- **settings.gradle** Se utiliza para configurar el proyecto en general.
- **gradle.properties** Archivo para definir diferentes variables que se pueden usar en nuestro proyecto de gradle. como la versión de una dependencia.
- **gradlew y gradlew.bat** Son dos ejecutables que permiten ejecutar Gradle en nuestro proyecto sin necesidad de tenerlo instalado globalmente.
- **src/** Contiene los directorios **main** y el **test**. En el directorio **main**, se encuentran las carpetas **java** y **res**, así como el archivo AndroidManifest.xml. En el directorio **test**, se incluyen las pruebas para verificar nuestro código.
- **.gradle/** Esta carpeta es creada por Gradle para almacenar y cachear las dependencias descargadas y otros códigos ya compilados, evitando tener que reconstruir todo desde cero cada vez.
- **build/** Contiene todas las clases y archivos que Gradle ha necesitado para compilar nuestro proyecto, incluyendo los archivos .jar



En el siguiente punto me centro en la perspectiva Android

Ficheros importantes en un proyecto Android con Gradle.

Perspectiva Android



- **AndroidManifest.xml.** Define la estructura y los componentes de la aplicación. Nombre del paquete, todas las activities de la aplicación..
- **Java/** Donde creamos las clases java y las clases de las activities donde se e define la lógica de los layout
- **res/** contiene todos los recursos de la aplicación, organizados en subdirectorios:
  - **drawable/** Contiene imágenes y gráficos que se se usan en la app pueden ser .png, .jpg o xml
  - **layout/** son los archivos xml que definen la estructura de la interfaz de usuario (ventanas app)
  - **mipmap/** se utiliza específicamente para almacenar íconos de la aplicación.
  - **Values/** Contiene archivos XML que almacenan recursos de valores.
    - **Strings.xml** cadenas de texto
    - **colors.xml** colores
    - **themes/** un archivo styles.xml donde defines los temas y estilos de tu aplicación.

Instalación de un emulador o dispositivo de pruebas.

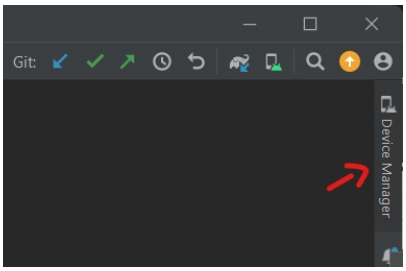
Emulador sirve para probar y depurar nuestras aplicaciones antes de publicarlas.

Hay tres formas de añadir un emulador a Android Studio:

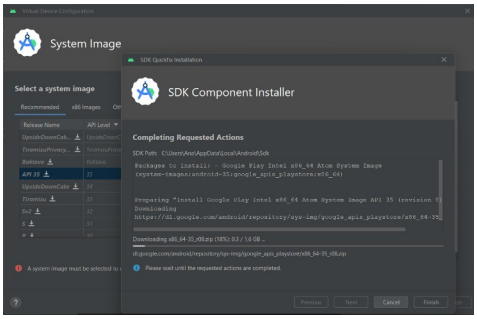
- Emulador Integrado
- Emulador de Terceros
- Dispositivo Físico

El emulador consume muchos recursos es aconsejable desarrollar utilizando un teléfono Android físico. (Yo utilizaré un Huawei Mate 20 Lite)

- Añadir un emulador virtual integrado AVD Manager

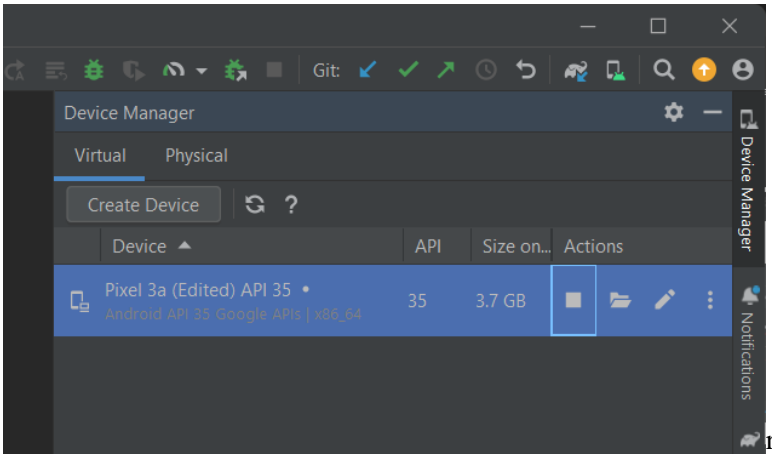


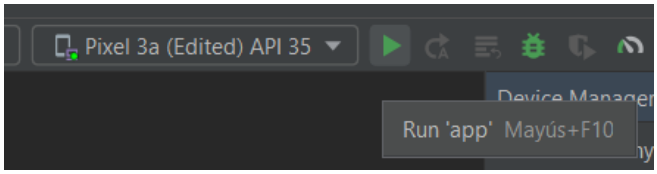
1. Hacemos clic en **Device Manager**
2. Clic en **Create Device**
3. **Elegimos un dispositivo** (en mi caso pixel 3a)
4. Luego elegimos una **API**. La descargamos y la seleccionamos (en mi caso API 35)
5. Le ponemos un **nombre** y elegimos la **orientación** del dispositivo Portrait o Landscape (en mi caso Portrait)



Nos aparecerá el dispositivo añadido.

Cuando ejecutemos el programa nos aparecerá el móvil. (Si tenemos más de uno tenemos que tenerlo seleccionado antes de ejecutar)





- Añadir un dispositivo Físico

Se puede conectar mediante USB o Wifi. Es recomendable conectarlo por USB

1. Habilitar en el teléfono las Opciones de Desarrollador

1. Ajustes del télefono
2. Acerca del teléfono

   Buscamos Numero de compilación

3. Numero de compilación (en mi caso **pulsamos encima 7 veces** y se **habilita el modo desarrollador**)

2. Habilitar Depuración USB
   1. Ajustes del teléfono
   2. Sistema (depende de que teléfono utilices). Puedes buscar en la lupa Opciones del desarrollador
   3. Habilitamos opciones de desarrollador
   4. Habilitamos Depuración USB
3. Conectar Dispositivo al pc
   1. Conectamos el móvil con un cable USB o tipo C

      Tener en cuenta:

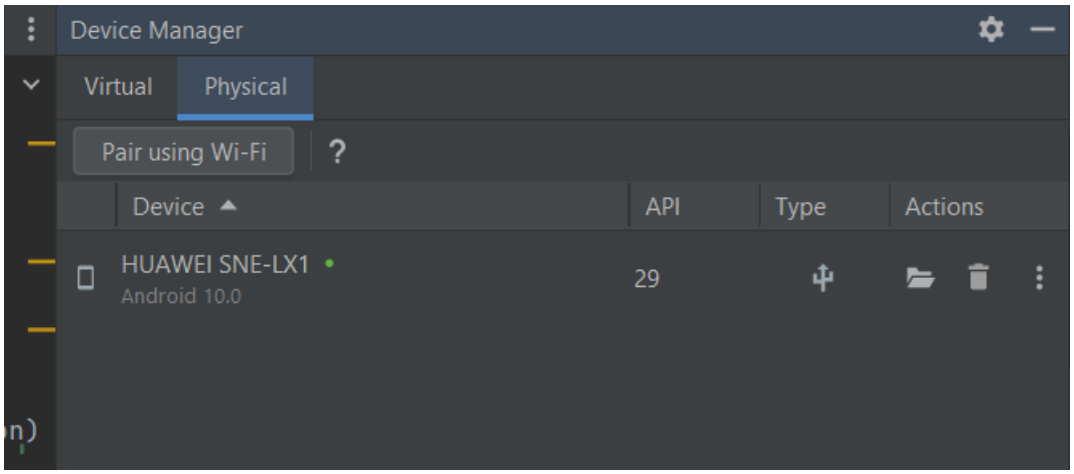
      1. El puerto no sea solo de carga.
      2. Tiene que servir para transmitir archivos. Funcione el puerto y el cable.
4. Seleccionar el modo de conexión en el dispositivo

   Al conectar el dispositivo elegir la opción T**ransferir Archivos** o **MTP**

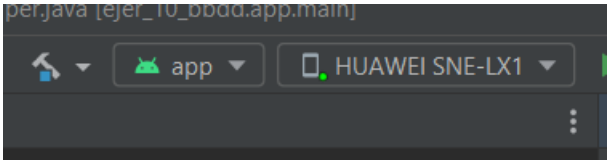
Tener en cuenta: Puede ser que algunos dispositivos menos conocidos necesiten controladores (podrás descargar en la web del fabricante) Normalmente no es necesario.

5. Abrir Android Studio

   en Physical debe aparecer el dispositivo



6. Antes de ejecutar la app hay que tener en cuenta que tenemos seleccionado el dispositivo físico



7. Al ejecutar

   La aplicación se instalará y ejecutará en el dispositivo

Concepto de activity, ficheros que contiene y cómo crear una nueva.

Activity es un componente que crea una única pantalla en la aplicación

La activity para que funcione tiene que estar declarada en AndroidManifest.xml (suele hacerse de forma automática al crearla)

Ficheros que contiene una Activity

MainActivity.java Controlará la lógica de esta pantalla. Todas las pantallas están controladas por un componente que se llama **Activity**. Al ser una clase Java, se empieza con mayúscula siguiendo la nomenclatura **UpperCamelCase**.

Nuuestra actividad XML asociada al fichero Java. Este XML contendrá todos los elementos visuales de la pantalla.

- **MainActivity.java** (Lógica programación con Java)
- **activity_main.xml** (Contendrá todos los elementos visuales)

Ciclo de vida de una Activity

1. **OnCreate()**
   - Se llama cuando se crea la actividad.
   - Se inicializan los componentes Asociamos el objeto al elemento correspondiente del xml por ejemplo: `login = findViewById(R.id.iniciarSesion);`
   - onClickListener de los botones
2. **onStart()**

   Esta llamada hace que la actividad sea visible para el usuario. Realizar acciones que deben comenzar cuando la actividad está en primer plano. Como iniciar animaciones o cargar datos.
3. **OnResume()**

   Se llama cuando la actividad comienza a interactuar con el usuario.

   Reanudar tareas que se detuvieron en onPause()
4. **onPause()**

Se llama cuando la actividad está a punto de ser interrumpida (por ejemplo, cuando otra actividad está en primer plano).

Guardar el estado y liberar recursos que no sean necesarios

5. **onStop()**

   Se llama cuando la actividad ya no es visible para el usuario.

6. **OnRestart()**

   Se llama cuando la actividad se está reiniciando después de haber estado detenida

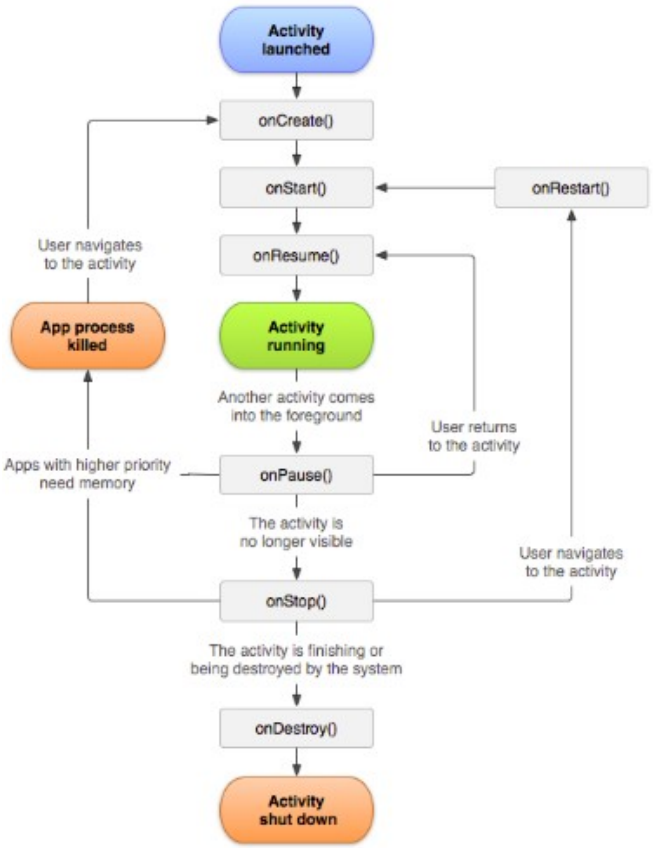
   Puedes volver a preparar la actividad para ser mostrada.

7. **onDestroy()**

   Se llama antes de que la actividad sea destruida.

   Aquí debes liberar todos los recursos utilizados durante la actividad.

Explicación Ciclo de Vida de una Activity



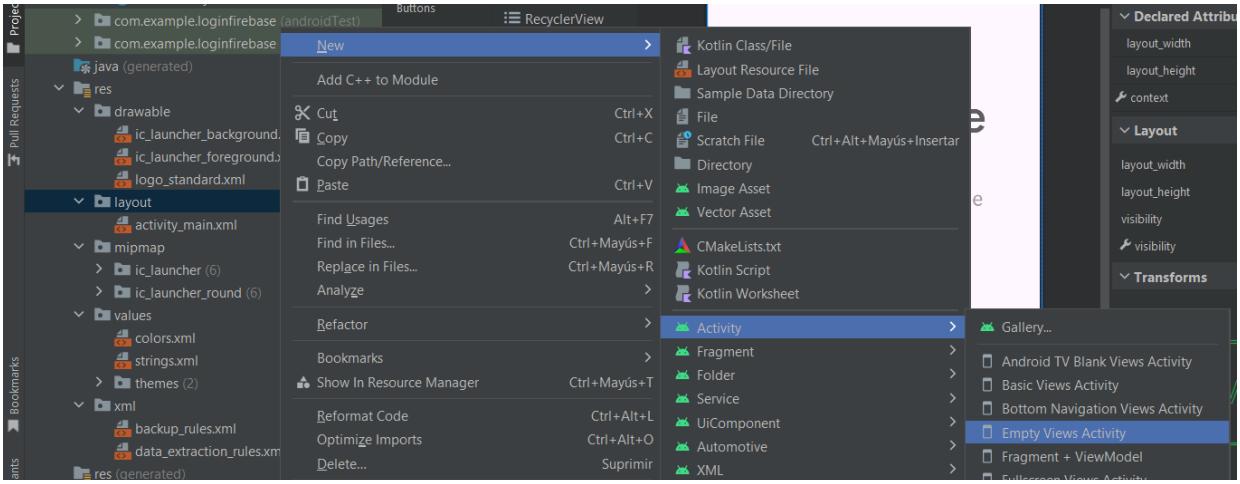
- 1. Cuando arrancamos la app se lanza el **primer activity** el que hayamos definido en el AndroidManifest.xml
- 2. Una vez se lanza la activity el primer método que se lanza es el **onCreate()** (solo la vez que se arranca la aplicación) después **inmediantamente** por **onStart() y onResume()**

  Una vez lanzado los **tres métodos anteriores** la **app esta en ejecución.** Entones ya puede pasar a un estado de pausa si la dejamos en segundo plano (abrimos otra)
- 3. **onPause()** si dejamos la aplicación en segundo plano (si llega un mensaje y abrimos el mensaje sobre esta app) y si volvemos a abrirla volvería al método onResume()
- 4. **onStop()** si le damos al botón inicio del móvil y se queda la app en pestañas

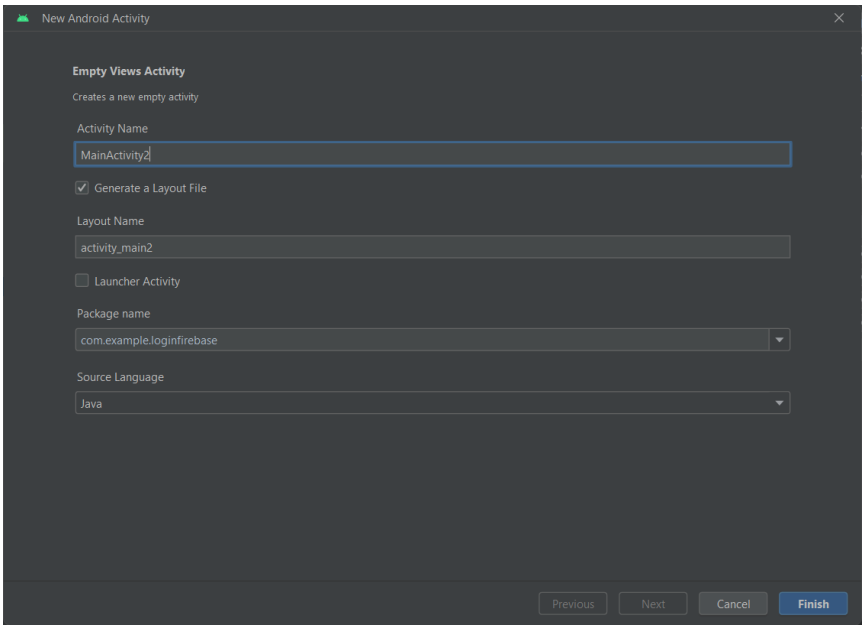
  El sistema operativo define cuando una app esta para o pausada
- **5. onDestroy()** la app se finaliza, este mucho tiempo parada, error irreparable...

Como crear una activity

Clic derecho en **Layout > New > Activity > Empty View Activity**



- **Activity Name** ponemos el nombre de la clase Java que controlará esta pantalla. Todas las pantallas están controladas por un componente que se llama Activity. Al ser una clase Java, se empieza con mayúscula siguiendo la nomenclatura **UpperCamelCase**.
- **Layout Name** ponemos el nombre que tendrá nuestra actividad XML asociada al fichero Java. Este XML contendrá todos los elementos visuales de la pantalla.



XML

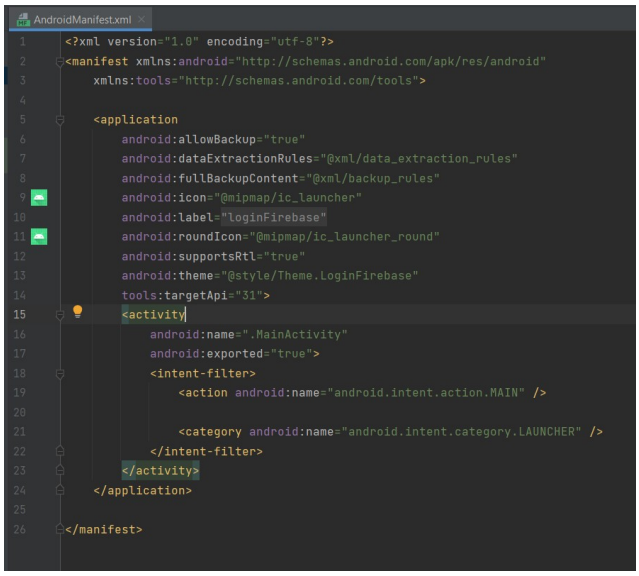
Interfaz en XML.

La estructura y el diseño de la interfaz o Activities se define en un documento xml. Permitiendo crear interfaces de usuario de manera más visual y organizada.

Otros XML importantes.

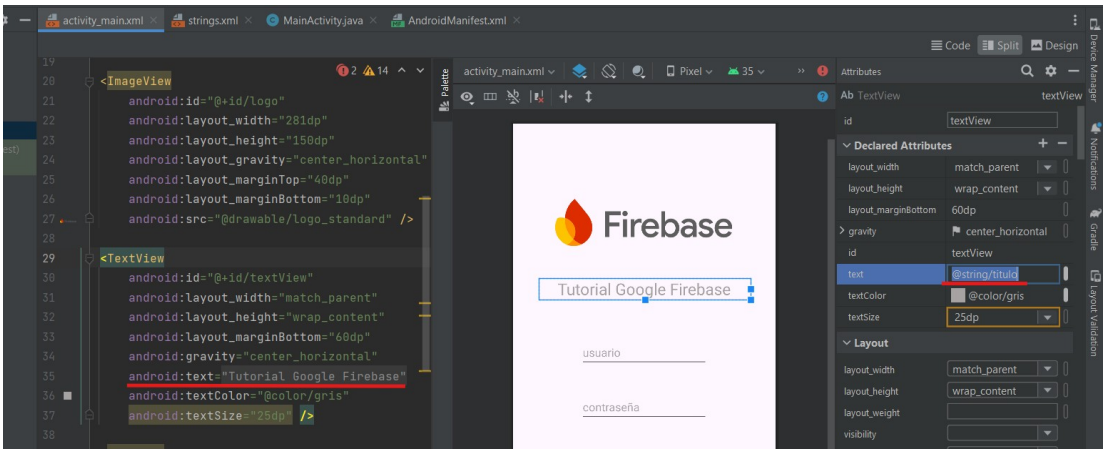
- **AndroidManifest.xml** Define la estructura y los componentes de la aplicación. Nombre del paquete, todas las activities de la aplicación.

  Si tenemos varias Activities pero no la tenemos declaradas aquí es como si no existieran en la aplicación realmente.



- **strings.xml** En este archivo xml vamos a definir las cadenas de texto que usará nuestra app

  En lugar de poner el titulo directamente (sería incorrecto, si necesitamos cambiarlo en todas las pantallas. Tendríamos que ir una por una). Creamos una etiqueta con atributo titulo en el xml. Y Le asignamos al textView el nombre del atributo @strings/titulo



- **colors.xml** Lo mismo que con strings pero para poner colores. En este archivo xml vamos a definir los colores que usará nuestra la aplicación.

  El elemento será color y agregaremos el atributo gris, este atributo funciona como el id.

  ```
  <color name="gris">#A8A3A3</color>
  ```

  En el xml del layout o desde la interfaz de Android Studio ponemos el TextView de color gris ponemos @color/gris

  ```
  android:textColor="@color/gris"
  ```

También podemos cambiar el color dándole al cuadrado de la izquierda

Java

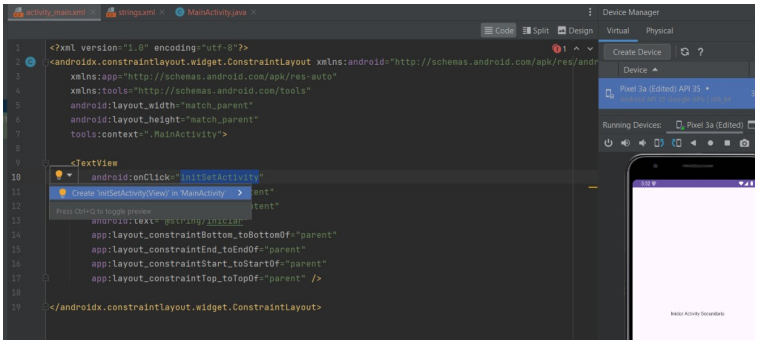
MainActivity.java Controlará la lógica de esta pantalla. Como se explica en este punto

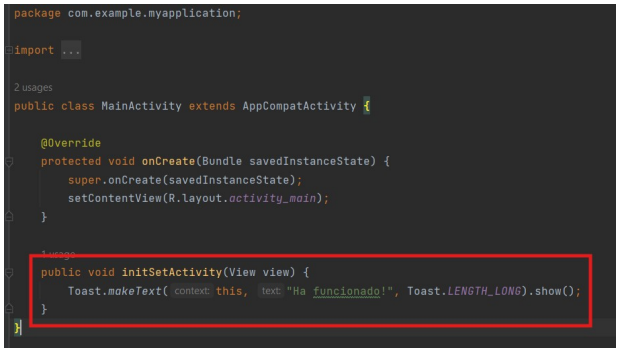
En este documento, dentro del método onCreate(), es donde tenemos que instanciar los objetos creados con los elementos que vamos a utilizar del XML. También es donde se crearán los métodos del ciclo de vida de la Activity.

Eventos de botón.

Para este ejemplo he creado un proyecto nuevo con dos activities. La activity principal con un TextView al hacerle clic

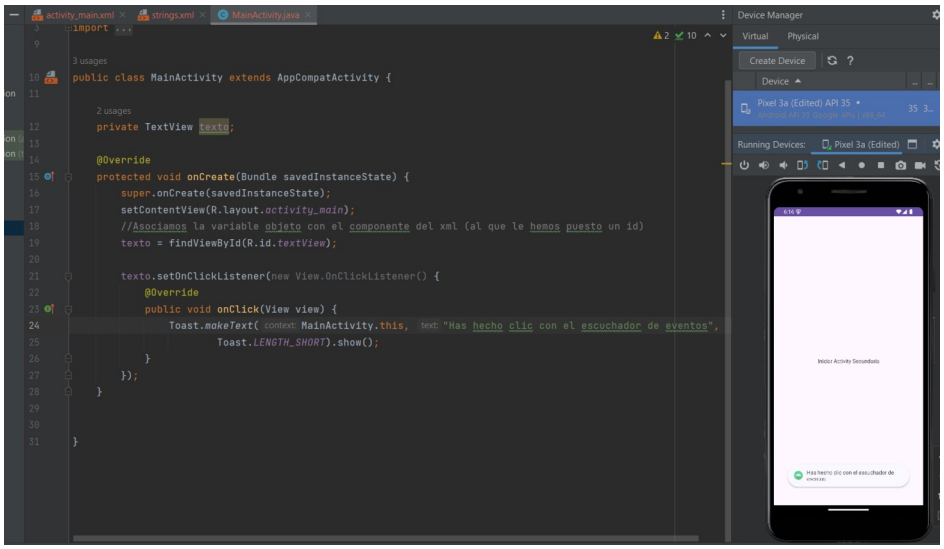
- **Forma 1** Añadir e**n el elemento del xml un onClick.** Le damos doble clic al nombre que le hemos puesto al método nos mostrara un mensaje (toast que hare dentro del método) y nos sale la bombillita para crearnos un método en la clase MainActivity.java



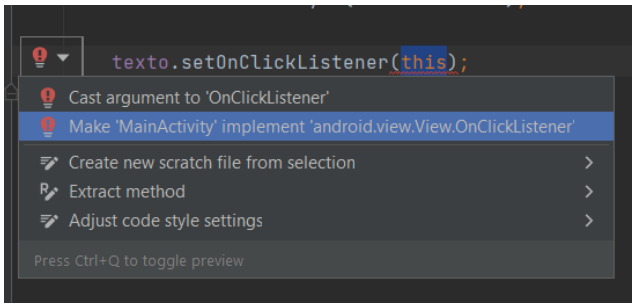


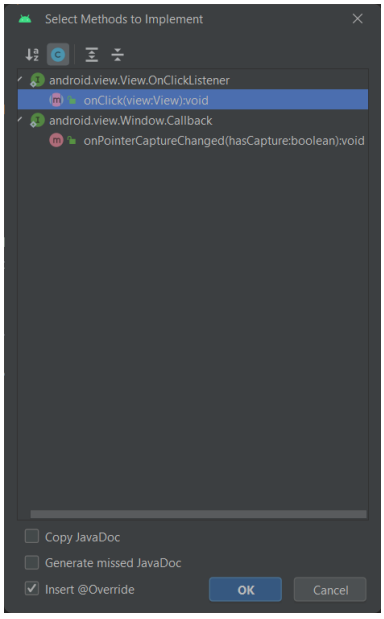
- **Forma 2 Definiendo la clase anónima**

  1. Poniéndole un id al textView.
  2. Creando una variable objeto.
  3. Instanciando el elemento xml
  4. Creando un método texto.setOnClickListener(new View.OnClickListener()



- **Forma 3** vamos a declarar en esta clase la opción de gestionar evento clic. Vamos a obligar que l**a clase implemente una interfaz**, al implementarla estaría obligada a **sobrescribir los métodos.**





El método ya no aparecería dentro del evento como antes, si no que aparece en un método a nivel de la clase.

De esta forma **si tengo más elementos que gestionar con el evento clic** en lugar de tener que hacer tantos bloques de código. Les hago referencia con todos los elementos que yo declare y todos apuntarian al public void onClick(View view). De manera que si hay varios elementos que apuntan al mismo onClic todos harían lo mismo. Lo que tendría que hacer un condicional if para saber desde que elemento han hecho clic y en cada caso haga algo distinto.

```
2 usages
private TextView texto;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //Asociamos la variable objeto con el componente del xml (al que le hemos puesto un id)
    texto = findViewById(R.id.textView);

    texto.setOnClickListener(this);
}


@Override
public void onClick(View view) {
    int id = view.getId();
    if(id == R.id.textView){
        Toast.makeText( context: this,  text: "implementado nivel clsse", Toast.LENGTH_SHORT).show();
    }else{
        //...
    }

}
}
```

La clase Intent:

- Abrir otra activity

  Intent es una clase que al crear un objeto de está he introduciendo dos parámetros que son la activity donde nos encontramos y a la que queremos ir. Se encarga del paso entre ellas.

  - Iniciar nuevas Activities
  - Pasar datos a través de activities con putExtra()
  - Iniciar servicios en segundo plano

1. **Intent explícito:** Iniciar una activity dentro de una misma app, indicando la activity donde queremos ir.

```
public class MainActivity extends AppCompatActivity {

    2 usages
    private Button boton;

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

        boton = findViewById(R.id.button);

        boton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //Código necesario para iniciar la siguiente activity
                //Intent explícito

                Intent abrirPantalla = new Intent( packageContext: MainActivity.this, MainActivity2.class);
                startActivity(abrirPantalla);
            }
        });
```

- con putExtra también **podemos** declarar parámetros y **pasarlos a la siguiente activity**

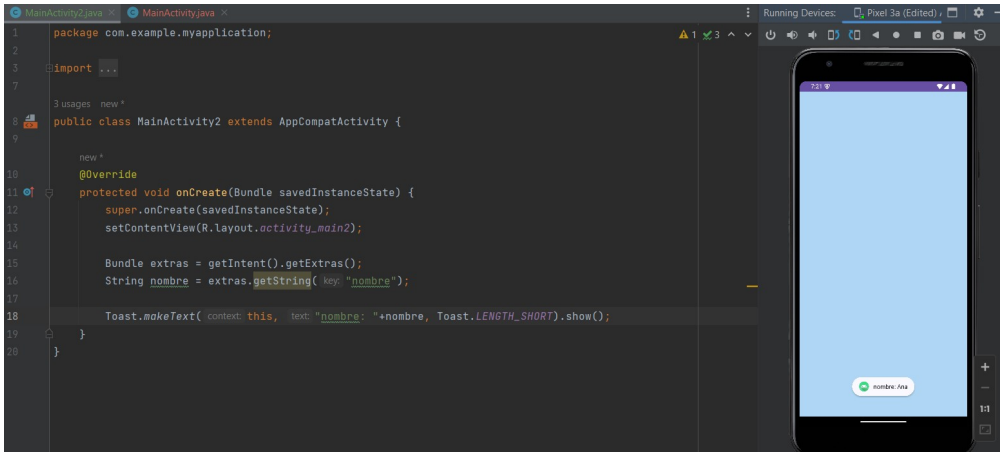
```
Intent abrirPantalla = new Intent( packageContext: MainActivity.this, MainActivity2.class);

abrirPantalla.putExtra( name: "nombre",  value: "Ana");

startActivity(abrirPantalla);
```

- para **recoger esos datos en la activity 2:** Podemos hacerlo con Intent y Bundle

  Usa Intent iniciar una nueva actividad y pasar algunos datos simples.

  Usa Bundle asar múltiples datos o datos más complejos.

  Obtener datos Activity Con Bundle:



Obtener datos Activity con Intent:

```
package com.example.myapplication;

import ...

3 usages  new *
public class MainActivity2 extends AppCompatActivity {

    new *
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        Intent intent = getIntent();
        String nombre = intent.getStringExtra( name: "nombre");

        Toast.makeText( context: this,  text: "nombre: "+nombre, Toast.LENGTH_SHORT).show();
    }
}
```

2. **Intents implícitos:** No una activity si no una acción genérica. Nos permite llamar al SO para que otra aplicación del dispotivo resuelva nuestra petición por ejemplo reproducción de un archivo de música busca una app que pueda reproducir mostrándonos una lista y el usuario eligiéndola. Por ejemplo hacer llamadas, alarmas..

- Abrir correo electrónico

  Una vez instanciado todo. Hacemos el onClick de cada botón.

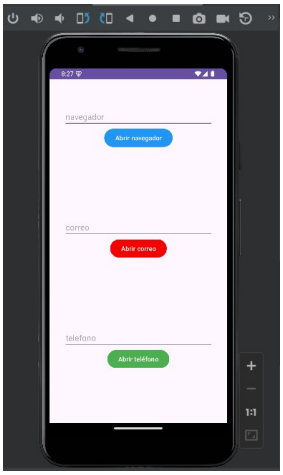
  - Obtenemos el **correo introducido** en el EditText con .getText().toString().trim();
  - Toast si el campo esta vacio
  - **Intent Implícito:** Intent.ACTION_SENDTO
    - crear un intent para enviar correo
    - Usamos Uri.parse("mailto: " + email) para pasar el número de teléfono a la app
    - Abrimos la app

```
botonTelefono.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Obtener el número de teléfono del campo de entrada
        String numeroTelefono = telefono.getText().toString().trim();

        // Validar que no esté vacío
        if (!numeroTelefono.isEmpty()) {
            Intent tlfIntent = new Intent(Intent.ACTION_DIAL);
            tlfIntent.setData(Uri.parse( uriString: "tel:" + numeroTelefono)); // Usar el número de teléfono ingresado
            startActivity(tlfIntent);
        } else {
            // Mostrar un mensaje si el campo está vacío
            Toast.makeText( context: MainActivity.this,  text: "Por favor, ingresa un número de teléfono", Toast.LENGTH_SHORT).show();
        }
    }
});
```

- Abrir teléfono
  - Obtenemos el numero teléfono introducido en el EditText con getText().toString().trim()
  - Toast si el campo esta vacio
  - **Intent Implícito:** Intent.ACTION_DIAL
    - crear un intent para abrir la app teléfono
    - Usamos Uri.parse("tel:" + numeroTelefono) para pasar el número de teléfono a la app
    - Abrimos la app

```java
botonCorreo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Obtener la dirección de correo del campo de entrada
        String email = correo.getText().toString().trim();

        // Validar que no esté vacío
        if (!email.isEmpty()) {
            Intent emailIntent = new Intent(Intent.ACTION_SENDTO);
            emailIntent.setData(Uri.parse( uriString: "mailto:" + email)); // Usar la dirección de correo ingresada

            try {
                startActivity(emailIntent);
            } catch (Exception e) {
                Toast.makeText( context: MainActivity.this,  text: "No hay aplicaciones de correo instaladas", Toast.LENGTH_SHORT).show();
            }
        } else {
            // Mostrar un mensaje si el campo está vacío
            Toast.makeText( context: MainActivity.this,  text: "Por favor, ingresa una dirección de correo", Toast.LENGTH_SHORT).show();
        }
    }
});
```

- Abrir navegador
  - Obtenemos url introducida
  - Toast si el campo esta vacio
  - **Intent Implícito:** Intent.ACTION_VIEW
    - crear un intent para abrir la url en el navegador
    - Usamos Uri.parse(urlTexto) para pasar la url a la app
    - Abrimos la app




```java
botonNav.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Obtener la URL del campo de entrada
        String urlTexto = url.getText().toString().trim();

        // Validar que no esté vacío
        if (!urlTexto.isEmpty()) {
            // Validar que la URL comienza con "http://" o "https://"

            Intent navIntent = new Intent(Intent.ACTION_VIEW);
            navIntent.setData(Uri.parse(urlTexto));
            try {
                startActivity(navIntent);
            } catch (Exception e) {
                Toast.makeText( context: MainActivity.this,  text: "No se pudo abrir el navegador", Toast.LENGTH_SHORT).show();
            }
        } else {
            // Mostrar un mensaje si el campo está vacío
            Toast.makeText( context: MainActivity.this,  text: "Por favor, ingresa una URL", Toast.LENGTH_SHORT).show();
        }
    }
});
```

SQLite

Base de datos embebida. La BDD sera un fichero dentro de nuestra aplicación.

Si se desinstala la aplicación se pierde la BDD.

Para la gestión de la BDD vamos a utilizar SQLite.

Para poder utilizar una **base de datos SQLite será necesario heredar** de la clase **SQLiteOpenHelper**

Está clase tendrá:

- **un constructor**
- **dos métodos abstractos**
  - **onCreate():** consultas tipo create table
  - **onUpgrade():** operaciones para actualizar la BDD, copia seguridad
  - **Método** para **cerrar la conexión a la BDD**

- **Insertar valores en las tablas**, usaremos el método **getWritableDatabase()** para obtener una referencia a la base de datos en modo escritura

  Una vez tengamos esa referencia. Ya podremos realizar todas las acciones. **Insertar datos** usamos el método **execSQL()** Le pasamos sentencias insert. Mediante el método put, podremos indicarle al objeto los valores a almacenar, indicando el nombre del atributo y su valor y **cerramos** la BDD con el **método close()**

- **Borrar valores BDD**

  usaremos también el método **getWritableDatabase()**

  eliminar datos podemos utilizar el método **execSQL(),** al que le podremos **pasar las sentencias DELETE**

  podremos pasarle nombre tabla, condiciones y el valor de las condiciones

- **Actualizar Valores BDD**

  También usaremos el método **getWritableDatabase()** obtener una referencia a la BDD en modo escritura .

  Para eliminar datos podemos utilizar el método **execSQL()** al que le pasamos las sentencias UPDATE

  Tenemos que psarle nombre, valores, condición o condiciones y valor de las condiciones

  Podemos usar los operadores AND Y OR.

  Cuando eliminamos valores, las interrogaciones no tendrán que tener comillas ni simples ni dobles.

- **Consultar valores BDD**

  para consultar valores debemos suar el método **getReadableDatabase()** para obente runa referencia a la BDD en modo lectura.

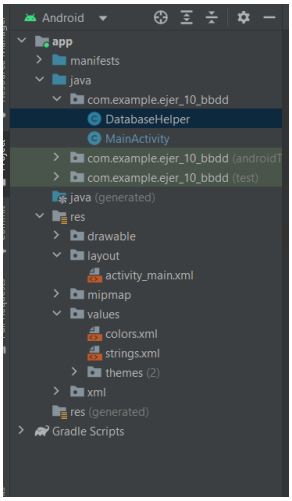
  - **SQLite** nos provee de la clase **Cursor**, nos permite seleccionar los valores de una tabla de una forma muy sencilla y sin tener que utilizar una sentencia SELECT de SQL.

- Y la función **Query**

le pasamos nombre, columnas, columas de la cláusula where y valores de la columna where, columna group by valores de la cláusula having y orden de la cláusula order by.

- Para obtener los valores tenemos los métodos **getString**, **getInt** a los que les tendremos que pasar el índice. Si queremos saber la cantidad de elementos recuperados por la consulta **getCount()**
- ejecutar una consulta SQL **con rawQuery, nos devolverá objeto Cursor**

Crear base de datos y tabla desde una clase de Java.



- **Creamos una clase** que **herede de SQLiteDataHelper**

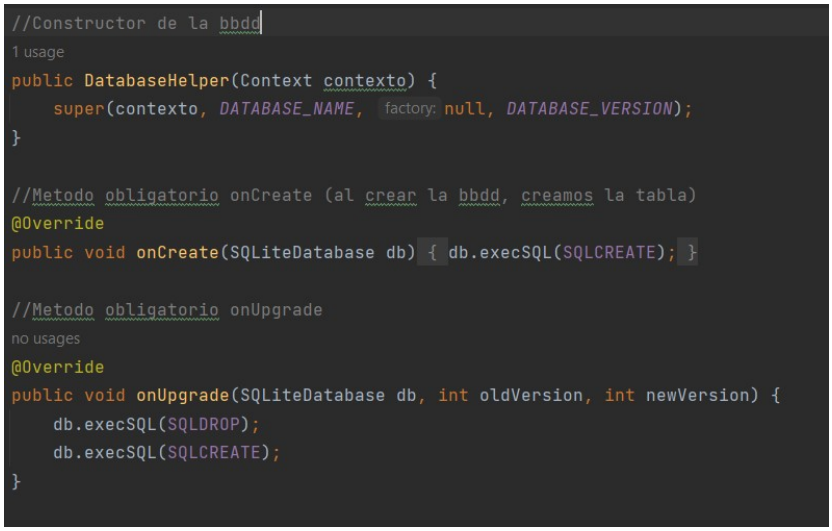
Dentro de la clase creamos los **atributos,** en este ejemplo:

- **SQLCREATE** Para crear la tabla Profesores con id, nombre y apellido
- **SQLDROP** Comando SQL para elimianr la tabla Profesores si ya existe
- **DATABASE_VERSION** Version de la base de datos
- **DATABASE_NAME** Nombre del archivo de la base de datos

```
public class
DatabaseHelper extends SQLiteOpenHelper {

    no usages
    private Context contexto;
    2 usages
    private final String SQLCREATE = "CREATE TABLE Profesores (id INTEGER PRIMARY KEY AUTOINCREMENT, nombre TEXT, apellido TEXT)";
    1 usage
    private final String SQLDROP = "DROP TABLE IF EXISTS Profesores";

    1 usage
    public static final int DATABASE_VERSION = 1;
    1 usage
    public static final String DATABASE_NAME = "DBProfesores.db";
```

Constructor:

- **El constructor llama al constructor de la clase padre SQLiteOpenHelper** (de la que heredamos)
  - **le pasamos los parámetros:**
    - **contexto** que es un objeto de la clase Contextque hemos creado donde los atributos
    - atributo **DATABASE_NAME** nombre BBD



```
//Constructor de la bbdd
1 usage
public DatabaseHelper(Context contexto) {
    super(contexto, DATABASE_NAME,  factory: null, DATABASE_VERSION);
}

//Metodo obligatorio onCreate (al crear la bbdd, creamos la tabla)
@Override
public void onCreate(SQLiteDatabase db) { db.execSQL(SQLCREATE); }

//Metodo obligatorio onUpgrade
no usages
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL(SQLDROP);
    db.execSQL(SQLCREATE);
}
```

    - **null** para el cursor factory (usa el predeterminado)
    - **DATABASE_VERSION**el atributo con la version de la BDD

- **onCreate**

  public void onCreate(SQLiteDatabase db) {

  db.execSQL(SQLCREATE);

  }

  Este método se ejecuta la primera vez que se crea la base de datos.

  Se crea la tabla profesores con excSQL y metiendo como parámetro el atributo que tiene la consulta de crear la tabla

- **onUpgrade**

  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

  db.execSQL(SQLDROP);

  db.execSQL(SQLCREATE);

  }

  este método se llama cuando se **detecta que la versión de la Base de datos se ha cambiado** por ejemplo de 1 a .2 Elimina la BBDD y luego la vuelve a crear. **Asegurar que la BBD este actualizada.**

Añadir y leer registros.

Añadir

Métodos personalizados: Cambiará según lo que queramos hacer en nuestra BBD

- **Método para insertar un profesor**
  - **Abre la base de datos en modo escritura** this.getWritableDatabase();
  - Usa **objeto** ContentValues para **almacenar** valores **nombre y apellido**
  - Inserta los valores en la tabla Profesores con el método insert
  - Cierra la conexión **de la base de datos**

```
Public void insertarProfesorEjemplo() {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues valores = new ContentValues();
    valores.put("nombre", "Juan");
    valores.put("apellido", "Perez");
    db.insert("Profesores", null, valores); db.close();
}
```

```
//////////////////// Metodos opcionales, segun necesidad ////////////////////////

//Metodo para insertar un profesor de ejemplo
1 usage
public void insertarProfesorEjemplo() {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues valores = new ContentValues();
    valores.put("nombre", "Juan");
    valores.put("apellido", "Perez");
    db.insert( table: "Profesores",  nullColumnHack: null, valores);

    db.close();
}


//Método para comprobar si un profesor esta en la tabla
1 usage
public boolean comprobarProfesor(String nombre, String apellido) {
    SQLiteDatabase db = this.getReadableDatabase();
    String query = "SELECT * FROM Profesores WHERE nombre = ? AND apellido = ?";
    Cursor cursor = db.rawQuery(query, new String[]{nombre, apellido});

    boolean existe = cursor.getCount() > 0;

    cursor.close();
    db.close();
    return existe;
}
}
```

Leer

- **Método para comprobar si un profesor existe**
  - **Abre la base de datos en modo lectura** getReadableDatabase
  - **Ejecuta consulta SQL creadonla y ejecutandola con Cursor** usa parametro ? **Evitar inyecciones SQL**
  - comprueba con getCount si la **consulta devuelve alguún registro**
  - **Cierra** el cursor y la base de datos

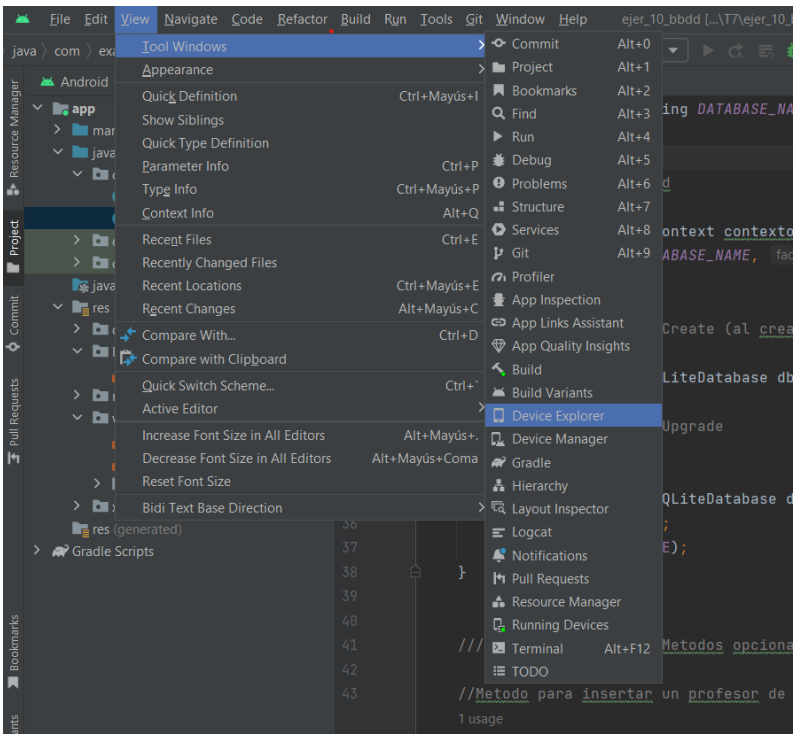
```
public boolean comprobarProfesor(String nombre, String apellido) {
    SQLiteDatabase db = this.getReadableDatabase();
    String query = "SELECT * FROM Profesores WHERE nombre = ? AND apellido = ?";
    Cursor cursor = db.rawQuery(query, new String[]{nombre, apellido});
    boolean existe = cursor.getCount() > 0;
    cursor.close(); db.close(); return existe;
    }
```

Cómo ver la tabla creada (donde está el fichero y cómo abrirlo).

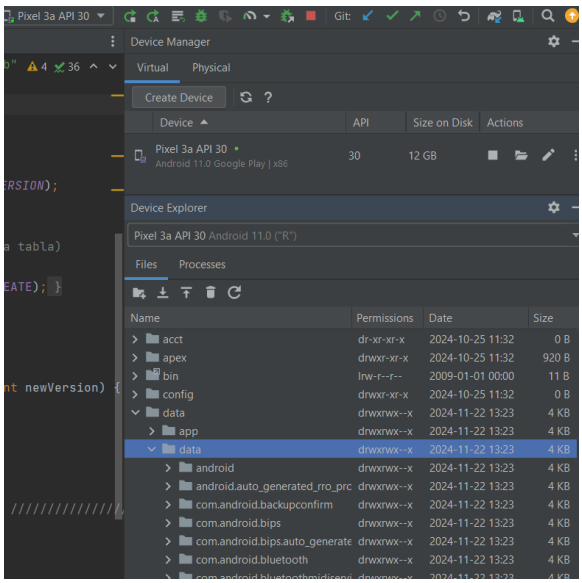
Usar el Device File Explorer en Android Studio

1. Hay que **tener ejecutada la aplicación en un emulador o dispositivo físico** y luego dale a **Menú**
2. **View > Tool Windows > Device File Explorer.**

   Luego abre una ventana donde se encuentran los ficheros del móvil emulado o el físico



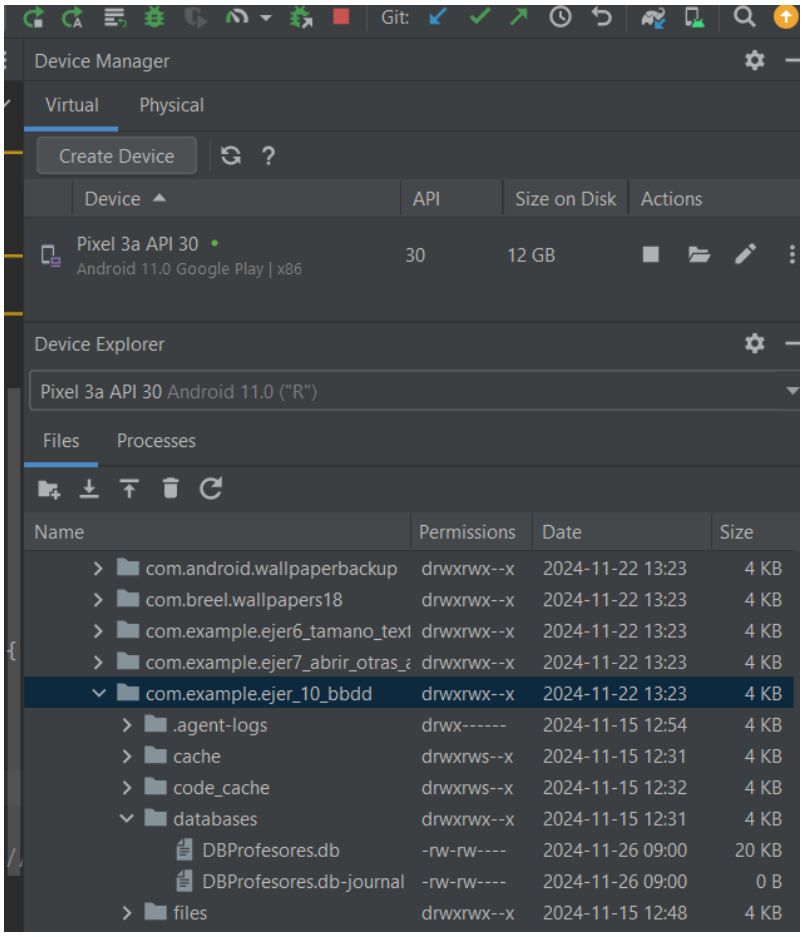
3. **Buscamos la carpeta data**



4. **Buscamos el nombre de nuestro proyecto en mi caso** ejer_10_bbdd

   Y en la carpeta databases que hay dentro encontraremos el **.db** la base de datos si ha funcionado su creación

   podemos descargarla y ver la BDD en **DB Browser for SQLite** o **SQLite CLI**



Backend*.