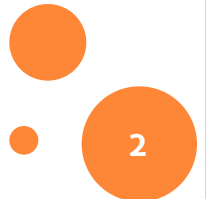


ALMACENAMIENTO DE INFORMACIÓN LOCAL



ÍNDICE

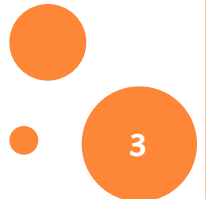
- Preferencias
- Ficheros
- Bases de datos locales



ALMACENAMIENTO DE DATOS

○ Preferencias

- Android permite almacenar las “preferencias” del usuario en la aplicación a través de la clase ***SharedPreferences***.
- Se utilizan para guardar pares del tipo ***clave-valor***
- Son persistentes
- Se puede guardar cualquier dato (de tipo primitivo) en ellas, pero se suelen usar para datos concretos y que no varíen



ALMACENAMIENTO DE DATOS

○ Preferencias

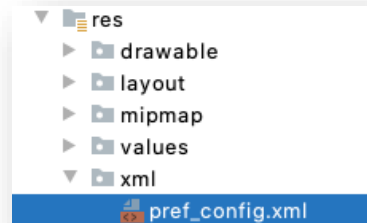
- Por defecto se trabaja sobre un único fichero que es público, pero se pueden crear (mediante código) otros públicos o privados
 - Desaconsejado por motivos de seguridad
- Más información en la documentación oficial:
 - <https://developer.android.com/guide/topics/ui/settings>



ALMACENAMIENTO DE DATOS

○ Preferencias

- Son necesarios 2 elementos:
 - Un fichero XML que contiene la configuración
 - Se puede crear dentro de res/xml



- Un **Fragment** de tipo **PreferenceFragmentCompat**
 - Será la interfaz para gestionar preferencias

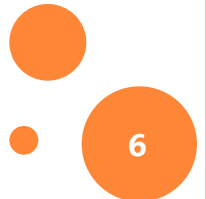


ALMACENAMIENTO DE DATOS

○ Preferencias

- Hay que añadir la dependencia en el gradle (fichero build.gradle del paquete app)

```
dependencies {  
    .....  
    implementation 'androidx.preference:preference:1.2.0'  
    .....  
}
```



ALMACENAMIENTO DE DATOS

○ Preferencias

- Crear una clase que herede de *PreferenceFragmentCompat*
- En su método *OnCreatePreferences(..)* se le indica cuál es el fichero XML donde están definidas las preferencias:

```
public class Preferencias extends PreferenceFragmentCompat {  
    @Override  
    public void onCreatePreferences(Bundle bundle, String s) {  
        addPreferencesFromResource(R.xml.pref_config);  
    }  
}
```

Fichero con la descripción de las preferencias

```
<?xml version="1.0" encoding="utf-8"?>  
<android.support.constraint.ConstraintLayout  
...  
    <fragment  
        android:id="@+id/fragment"  
        android:name="com.example.tema8ejercicio1.Preferencias"  
        .../>  
</android.support.constraint.ConstraintLayout>
```

En el layout de la actividad



ALMACENAMIENTO DE DATOS

○ Preferencias

- El fichero XML definirá qué preferencias se desean almacenar, su tipo, sus posibles valores, etc.

- El nodo raíz es **PreferenceScreen**
- Se pueden definir categorías con **PreferenceCategory**
 - El atributo **app:title** permite poner nombre a la categoría
- Tipos de preferencia
 - **SwitchPreference**: activar o desactivar (modo interruptor)
 - **CheckBoxPreference**: activar o desactivar (modo checkbox)
 - **EditTextPreference**: introducir un valor
 - **ListPreference**: elegir un valor de una lista
 - **MultiSelectListPreference**: elegir múltiples valores de una lista
- Atributos
 - **app:key** → clave para acceder a la preferencia
 - **app:title** → nombre de la preferencia
 - **app:summary** → descripción de la preferencia
 - ...

Se puede editar
gráficamente



ALMACENAMIENTO DE DATOS

○ Preferencias

```
<PreferenceScreen
    xmlns:app="http://schemas.android.com/apk/res-auto">

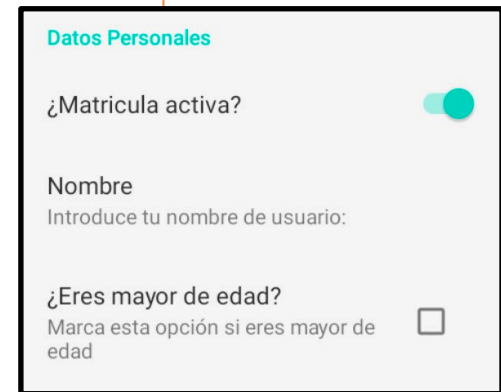
    <PreferenceCategory
        app:key="cat_datos_personales"
        app:title="Datos Personales">

        <SwitchPreferenceCompat
            app:key="switch"
            app:title="¿Matricula activa?"/>

        <EditTextPreference
            app:key="nombre"
            app:title="Nombre"
            app:summary="Introduce tu nombre de usuario:"
            app:dialogTitle="Introduce un nombre:"
            app:dialogIcon="@mipmap/ic_launcher" />

        <CheckBoxPreference
            app:key="mayor"
            app:title="¿Eres mayor de edad?"
            app:summary="Marca esta opción si eres mayor de edad" />

    </PreferenceCategory>
</PreferenceScreen>
```

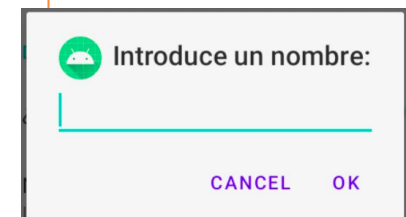



Datos Personales

¿Matricula activa? ☒

Nombre
Introduce tu nombre de usuario:

¿Eres mayor de edad?
Marca esta opción si eres mayor de edad ☐



 **Introduce un nombre:**

CANCEL OK



ALMACENAMIENTO DE DATOS

○ Preferencias

Están definidos en un fichero xml

Los valores que se muestran

Los valores que se recogen

```
<ListPreference
    app:dialogTitle="Elige una asignatura"
    app:entries="@array/asignaturas"
    app:entryValues="@array/codigosasignaturas"
    app:key="asigpref"
    app:summary="Elige tu asignatura preferida"
    app:title="¿Cual es tu asignatura preferida?" />

<MultiSelectListPreference
    app:dialogTitle="Elige varias asignaturas"
    app:entries="@array/asignaturas"
    app:entryValues="@array/codigosasignaturas"
    app:key="asigcurs"
    app:summary="Elige que asignaturas has cursado"
    app:title="¿Que asignaturas has cursado?" />
</PreferenceCategory>
```

Datos Academicos

¿Cual es tu asignatura preferida?

Elige tu asignatura preferida

¿Que asignaturas has cursado?

Elige que asignaturas has cursado

Elige una asignatura

- ☐ Desarrollo Avanzado de Software
- ☐ Administración de Sistemas
- ☐ Administración de Bases de Datos

CANCEL

Elige varias asignaturas

- ☐ Desarrollo Avanzado de Software
- ☐ Administración de Sistemas
- ☐ Administración de Bases de Datos

CANCEL OK



ALMACENAMIENTO DE DATOS

○ Preferencias

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="asignaturas">
    <item>Desarrollo Avanzado de Software</item>
    <item>Administración de Sistemas</item>
    <item>Administración de Bases de Datos</item>
  </string-array>
  <string-array name="codigosasignaturas">
    <item>DAS</item>
    <item>AS</item>
    <item>ABD</item>
  </string-array>
</resources>
```

En el interior de un fichero xml creado en la carpeta values. Por ejemplo en datos.xml



ALMACENAMIENTO DE DATOS

○ Preferencias

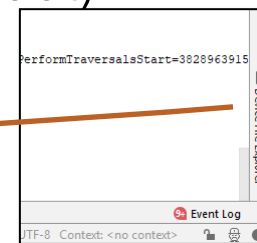
- El fichero con las preferencias se almacena en la ruta

*data/data/**nombrepaqueteaplicación**/shared_prefs/**nombrepaqueteaplicación**_preferences.xml*

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <boolean name="mayor" value="true" />
  <set name="asigcurs">
    <string>AS</string>
    <string>ABD</string>
  </set>
  <string name="asigpref">DAS</string>
  <string name="nombre">Unai</string>
  <boolean name="notifications" value="true" />
  <boolean name="switch" value="true" />
</map>
```

- Para acceder al fichero en un dispositivo real (sin conectar a Android Studio) hay que ser administrador (root)

En un dispositivo conectado o en un emulador, se puede acceder a través del Explorador de ficheros del dispositivo (esquina inferior derecha de Android Studio)



ALMACENAMIENTO DE DATOS

○ Preferencias

- Para acceder al fichero por defecto de *SharedPreferences*, usamos el método *getDefaultSharedPreferences(..)*

contexto

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
```

- Para leer datos desde un objeto *SharedPreferences*:
 - *getString* ("clave", valorpordefecto)
 - *getBoolean* ("clave", valorpordefecto)
 - *getInt* ("clave", valorpordefecto)
 - *getFloat* ("clave", valorpordefecto)
 - *getLong* ("clave", valorpordefecto)
 - *getStringSet* ("clave", valorpordefecto)



ALMACENAMIENTO DE DATOS

○ Preferencias

- Para leer datos desde un objeto *SharedPreferences*:

```
Boolean mayordeedad = prefs.getBoolean("mayor", false);
```

Valor por defecto

```
String asignaturapreferida = prefs.getString("asigpref", "DAS");
```

Nombre de la preferencia

```
if (prefs.contains("asigpref")) {  
    String asig01 = prefs.getString("asigpref", null);  
}
```

Cuando puede tener varios valores se usa la clase *Set*

Como he comprobado que existe no necesito valor por defecto

```
Set <String> d = prefs.getStringSet("asigcurs", null);  
if (d!=null) {  
    for (String str : d) {  
        ...  
    }  
}
```



ALMACENAMIENTO DE DATOS

○ Preferencias

- Para escribir datos en un objeto *SharedPreferences* se usa la clase *SharedPreferences.Editor* y sus métodos:

- *putString* ("clave", valor)
- *putBoolean* ("clave", valor)
- *putInt* ("clave", valor)
- *putFloat* ("clave", valor)
- *putLong* ("clave", valor)
- *putStringSet* ("clave", valor)

Si no existe esa clave, la crean

Objeto SharedPreferences

```
SharedPreferences.Editor editor= prefs.edit();  
editor.putString("asigpref", "DAS");
```

- Hay que aplicar los cambios

Síncrono, paraliza la ejecución y devuelve si se ha realizado correctamente

Asíncrono, no devuelve nada

```
editor.apply();
```

```
Boolean resultado= editor.commit();
```



ALMACENAMIENTO DE DATOS

○ Preferencias

- Para crear/acceder a un fichero de *SharedPreferences* distinto al de por defecto, se usa el método *getSharedPreferences(..)*

```
SharedPreferences prefs_especiales= getSharedPreferences("preferencias_especiales",  
Context.MODE_PRIVATE);
```

Nombre del fichero

Modo de acceso. Se recomienda
privado siempre

- La lectura y escritura de los valores es igual que con el fichero por defecto

```
SharedPreferences.Editor editor2= prefs_especiales.edit();  
editor2.putString("otrapreferencia", "nuevovalor");  
editor2.apply();
```

```
String otra = prefs_especiales.getString("otracosa", null);
```



ALMACENAMIENTO DE DATOS

○ Preferencias

- Para detectar cambios en el fichero de preferencias se usa el listener *OnSharedPreferencesChangeListener* en la clase que extiende a *PreferenceFragmentCompat*

```
public class Preferencias extends PreferenceFragmentCompat
    implements SharedPreferences.OnSharedPreferenceChangeListener {

    @Override
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {
        addPreferencesFromResource(R.xml.pref_config);
    }

    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String s) {
        switch (s) {
            case "nombre":
                ...
                break;
            default:
                break;
        }
    }
}
```



ALMACENAMIENTO DE DATOS

○ Preferencias

- El ciclo del vida del listener no se gestiona correctamente
- Debemos gestionarlo nosotros registrándolo y desregistrándolo en cada ocasión

```
public class Preferencias extends PreferenceFragmentCompat
    implements SharedPreferences.OnSharedPreferenceChangeListener {
    ...
    @Override
    public void onResume() {
        super.onResume();
        getPreferenceManager().getSharedPreferences().registerOnSharedPreferenceChangeListener(this);
    }

    @Override
    public void onPause() {
        super.onPause();
        getPreferenceManager().getSharedPreferences().unregisterOnSharedPreferenceChangeListener(this);
    }
}
```

Registrar

Desregistrar



ALMACENAMIENTO DE DATOS

○ Ejercicio 1:

- Crear una aplicación que almacene en las preferencias un nombre de usuario y le pregunte al usuario por su color de fondo preferido (en base a una lista).
- Haced que cada vez que se abra la aplicación se le muestre un mensaje de bienvenida usando el nombre de usuario y que las actividades tengan por defecto el fondo de su color preferido.



ALMACENAMIENTO DE DATOS

○ Ficheros

- Tres tipos de ficheros:
 - Incluidos en la aplicación (de sólo lectura)
 - Externos (de lectura y escritura)
 - En la memoria interna
 - Externos (de lectura y escritura)
 - En la memoria externa
 - Acceso mediante `getExternalFilesDir()`
 - Acceso mediante Storage Access Framework



ALMACENAMIENTO DE DATOS

○ Ficheros

- Los incluidos en la aplicación se almacenan en la carpeta *res/raw*
- Se leen mediante un objeto *InputStream* (de java.io)
- El nombre del fichero debe:
 - Contener solo minúsculas y _
 - Tener extensión

```
InputStream fich = getResources().openRawResource(R.raw.nombres);  
BufferedReader buff = new BufferedReader(new InputStreamReader(fich));  
try {  
    String linea = buff.readLine();  
    ...  
    fich.close();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Nombre del fichero,
sin extensión



ALMACENAMIENTO DE DATOS

○ Ficheros

- Para escribir un fichero externo en la memoria interna se usa la operación **openFileOutput**
- El fichero se almacena en la ruta
data/data/nombrepaqueteaplicación/files/

```
try {  
    OutputStreamWriter fichero = new OutputStreamWriter(openFileOutput("nombrefichero.txt",  
                                                                    Context.MODE_PRIVATE));  
    fichero.write("Estoy escribiendo en el fichero");  
    fichero.close();  
} catch (IOException e) {...}
```

Modo de acceso

- Soporta distintos modos de acceso:
 - MODE_PRIVATE : Sólo la aplicación que lo ha creado puede acceder a él.
 - MODE_APPEND : Si el fichero ya existía, añade la nueva información al final.



ALMACENAMIENTO DE DATOS

○ Ficheros

- Para leer un fichero externo en la memoria interna se usa la operación *openFileInput*

```
try {  
    BufferedReader ficherointerno = new BufferedReader(new InputStreamReader(  
                                                    openFileInput("nombrefichero.txt")));  
    String Linea = ficherointerno.readLine();  
    ficherointerno.close();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Nombre del fichero,
CON extensión



ALMACENAMIENTO DE DATOS

○ Ficheros

- Para trabajar con ficheros en la memoria externa es conveniente siempre asegurarse que hay memoria externa y si está accesible o no

```
boolean disponible, escritura;

String estado = Environment.getExternalStorageState();
if (estado.equals(Environment.MEDIA_MOUNTED))
{
    disponible = true;
    escritura = true;
}
else if (estado.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
    disponible = true;
    escritura= false;
}
else {
    disponible = false;
    escritura = false;
}
```



ALMACENAMIENTO DE DATOS

○ Ficheros

- Para escribir un fichero externo en la memoria externa

```
File path = //ELEGIR EL PATH - VER SIGUIENTES DIAPOSITIVAS
File f = new File(path.getAbsolutePath(), "nombreficheroexterno.txt");
Log.i("FICH", "PATH:" + path.getAbsolutePath());
try {
    OutputStreamWriter ficheroexterno = new OutputStreamWriter(new FileOutputStream(f));
    ficheroexterno.write("Estoy escribiendo en el ficheroexterno");
    ficheroexterno.close();
} catch (IOException e) {
    e.printStackTrace();
}
```



ALMACENAMIENTO DE DATOS

○ Ficheros

- Para leer un fichero externo de la memoria externa

```
File path = //ELEGIR EL PATH - VER SIGUIENTE DIAPOSITIVA
File f = new File(path.getAbsolutePath(), "nombreficheroexterno.txt");
try {
    BufferedReader ficheroexterno = new BufferedReader(new InputStreamReader(
                                                                    new FileInputStream(f)));
    String Linea= ficheroexterno.readLine();
    ficheroexterno.close();
} catch (IOException e) {
    e.printStackTrace();
}
```



ALMACENAMIENTO DE DATOS

○ Ficheros

- Para trabajar con ficheros en la memoria externa, hay que elegir el path en el que se quieren almacenar los ficheros

`this.getExternalFilesDir(null);`

Raíz de ficheros externos para la aplicación

- A partir de la API 29 (Android 10) el acceso programático a memoria externa no asignado a la aplicación está más controlado

`Environment.getExternalStorageDirectory();`

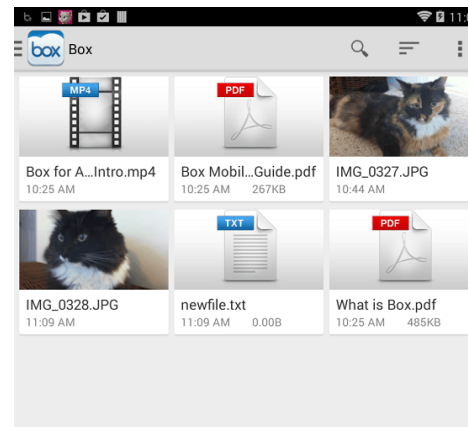
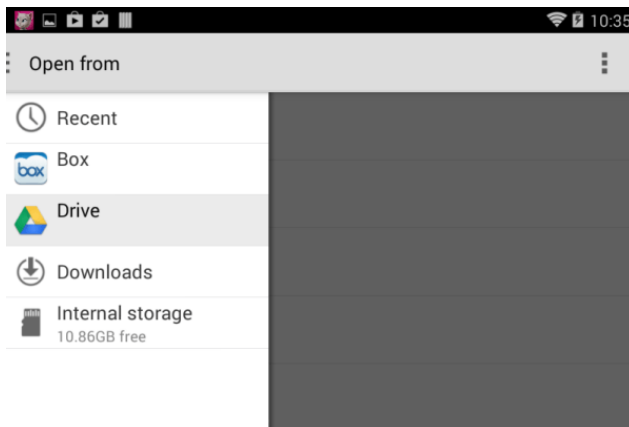
El uso de esta función está desaconsejado (*deprecated*)



ALMACENAMIENTO DE DATOS

○ Ficheros

- Storage Access Framework
- Muestra un selector de archivos al usuario
 - Unifica diferentes ubicaciones posibles para los archivos



ALMACENAMIENTO DE DATOS

○ Ficheros

- Storage Access Framework
 - *Crear nuevo fichero*
 - *Definir un `ActivityResultLauncher<Intent>`*
 - *Crear Intent implícito para llamar al selector de ficheros*

```
Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);  
intent.addCategory(Intent.CATEGORY_OPENABLE);  
intent.setType("text/plain");  
intent.putExtra(Intent.EXTRA_TITLE, "nuevo_fichero.txt");  
documentPickerLauncher.launch(intent);
```



ALMACENAMIENTO DE DATOS

○ Ficheros

- Storage Access Framework
 - *Crear nuevo fichero - recoger ruta al fichero seleccionado y procesar*

```
documentPickerLauncher = registerForActivityResult(  
    new ActivityResultContracts.StartActivityForResult(),  
    result -> {  
        if (result.getResultCode() == RESULT_OK) {  
            Intent data = result.getData();  
            if (data != null) {  
                Uri uri = data.getData();  
                writeToFile(uri);  
            }  
        }  
    })  
);
```



ALMACENAMIENTO DE DATOS

○ Ficheros

- Storage Access Framework
 - *Crear nuevo fichero - recoger ruta al fichero seleccionado y procesar*

```
private void writeToFile(Uri uri) {
    try {
        // Abre un OutputStream para escribir en el archivo
        getContentResolver().takePersistableUriPermission(
            uri, Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
        OutputStream outputStream = getContentResolver().openOutputStream(uri);
        if (outputStream != null) {
            String content = "Texto escrito en el archivo.";
            outputStream.write(content.getBytes());
            outputStream.close();
            Log.d("SAF", "Texto escrito correctamente en el archivo.");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



ALMACENAMIENTO DE DATOS

○ Ficheros

- Storage Access Framework
 - *Leer fichero*
 - *Definir launcher*
 - *Crear Intent implícito para llamar al selector de ficheros*

```
Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);  
intent.addCategory(Intent.CATEGORY_OPENABLE);  
intent.setType("text/plain"); //Podría ser otro tipo MIME como "image/*"  
documentReaderLauncher.launch(intent);
```



ALMACENAMIENTO DE DATOS

○ Ficheros

- Storage Access Framework
 - *Leer fichero - recoger ruta al fichero seleccionado y leer*

```
documentReaderLauncher = registerForActivityResult(  
    new ActivityResultContracts.StartActivityForResult(),  
    result -> {  
        if (result.getResultCode() == RESULT_OK) {  
            Intent data = result.getData();  
            if (data != null) {  
                Uri uri = data.getData();  
                if (uri != null) {  
                    readTextFile(uri);  
                }  
            }  
        }  
    })  
);
```



ALMACENAMIENTO DE DATOS

○ Ficheros

- Storage Access Framework
 - *Leer fichero - recoger ruta al fichero seleccionado y leer*

```
private void readTextFile(Uri uri) {  
  
    try {  
        InputStream inputStream = getContentResolver().openInputStream(uri);  
        if (inputStream != null) {  
            BufferedReader reader = new BufferedReader(new  
InputStreamReader(inputStream));  
            String line;  
            while ((line = reader.readLine()) != null) {  
                // Procesar cada línea del archivo  
                // ...  
            }  
            reader.close();  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```



ALMACENAMIENTO DE DATOS

- Ejercicio 2:
 - Crear una aplicación con 2 actividades:
 - Una de ellas muestra las palabras contenidas en un fichero
 - La otra permite incluir más palabras al fichero.



ALMACENAMIENTO DE DATOS

○ Bases de datos locales

- Android incluye soporte para Bases de Datos **SQLite**
- Se usan clases que extiendan **SQLiteOpenHelper**
- El constructor de la clase recibe los siguientes parámetros
 - **Contexto**: el contexto usado
 - **Nombre**: nombre de la BD
 - **CursorFactory**: normalmente será **null**. Para cuando se usen clases **Cursor** personalizadas (deben heredar de **Cursor**)
 - **Versión**: el número de versión de la BD

```
public class miBD extends SQLiteOpenHelper {  
  
    public miBD(@Nullable Context context, @Nullable String name,  
                @Nullable SQLiteDatabase.CursorFactory factory, int version) {  
        super(context, name, factory, version);  
    }  
    ...  
}
```



ALMACENAMIENTO DE DATOS

- Bases de datos locales
 - Además hay que sobrescribir dos métodos
 - *onCreate (...)*: se ejecuta cuando hay que crear la BD porque no existe.
 - *onUpgrade (...)*: se ejecuta cuando la versión de la BD que queremos usar y la que existe no coinciden.
 - Para ejecutar sentencias SQL se usan objetos de tipo *SQLiteDatabase* y el método *execSQL (String)*

```
public void onCreate(SQLiteDatabase sqLiteDatabase) {  
    sqLiteDatabase.execSQL("CREATE TABLE Usuarios ('Codigo' INTEGER PRIMARY KEY  
        AUTOINCREMENT NOT NULL, 'Nombre' VARCHAR(255))");  
}
```



ALMACENAMIENTO DE DATOS

- Bases de datos locales
 - La clase ***SQLiteOpenHelper*** (y las que la extienden), proporcionan dos formas de obtener un objeto de tipo ***SQLiteDatabase***
 - ***getReadableDatabase()***: En modo sólo lectura
 - ***getWritableDatabase()***: En modo lectura/escritura

```
SQLiteDatabase bd = getWritableDatabase();
```

Si lo hacemos en la propia
SQLiteOpenHelper

```
miBD GestorDB = new miBD (this, "NombreBD", null, 1);  
SQLiteDatabase bd = GestorDB.getWritableDatabase();
```

Si lo hacemos desde otra clase



ALMACENAMIENTO DE DATOS

- Bases de datos locales
 - Para hacer *INSERT*, *UPDATE* y *DELETE*
 - Usando el método *execSQL()*

```
bd.execSQL("INSERT INTO Estudiantes ('Nombre') VALUES ('Mikel')");  
bd.execSQL("UPDATE Estudiantes SET Nombre='Andoni' WHERE Codigo=2");  
bd.execSQL("DELETE FROM Estudiantes WHERE Nombre='Alicia' ");
```

- Usando los métodos concretos

- *insert (nombretabla, columnanull, values)*

Qué valor asignar a las columnas a las que no se les da valor específico

Pares de valores a añadir

```
ContentValues nuevo = new ContentValues();  
nuevo.put("Nombre", "Unai");  
bd.insert("Estudiantes", null, nuevo);
```



ALMACENAMIENTO DE DATOS

- Bases de datos locales

- Para hacer **INSERT**, **UPDATE** y **DELETE**

- Usando los métodos concretos

- **update** (*nombretabla*, *values*, *clausulawhere*, *argumentos*)

Pares de valores a modificar

La clausula WHERE del SQL

Argumentos para la clausula

```
ContentValues modificacion = new ContentValues();  
modificacion.put("Nombre", "Antonio");  
bd.update("Estudiantes", modificacion, "Codigo=2", null);
```

- Los **argumentos** sirven para usar valores variables en la clausula WHERE:

- Tienen forma de **array** de strings
 - Se sustituyen por el símbolo '?', en el mismo orden:

```
String[] argumentos = new String[] {"Mikel", "2"};  
bd.update("Estudiantes", modificacion, "nombre=? AND Codigo=?", argumentos);
```



ALMACENAMIENTO DE DATOS

- Bases de datos locales
 - Para hacer *INSERT*, *UPDATE* y *DELETE*
 - Usando los métodos concretos
 - *delete (nombretabla, clausulawhere, argumentos)*

La clausula WHERE del SQL

Argumentos para la clausula

```
bd.delete("Estudiantes", "nombre='mikel'", null);
```



ALMACENAMIENTO DE DATOS

- Bases de datos locales
 - Para hacer ***SELECT***
 - El resultado se recoge en un objeto de tipo ***Cursor***
 - Permite métodos para navegar por el resultado y extraer los datos:
 - *moveToFirst()*
 - *moveToNext()*
 - *getInt(numcolumna)*
 - *getFloat(numcolumna)*
 - *getString(numcolumna)*
 -
 - Dos formas de ejecutar la consulta desde un objeto de tipo ***SQLiteDatabase***:
 - *rawQuery(consulta, argumentos)*
 - *query (nombredetabla, campos, clausulawhere, argumentos, clausulagroupby, clausulahaving, clausulaorderby)*



ALMACENAMIENTO DE DATOS

- Bases de datos locales
 - Para hacer *SELECT*

```
Cursor c = bd.rawQuery("SELECT Codigo,Nombre FROM Estudiantes WHERE Codigo > 2", null);
```

```
String[] campos = new String[] {"Codigo", "Nombre"};  
String[] argumentos = new String[] {"2"};  
Cursor cu = bd.query("Estudiantes", campos, "Codigo>", argumentos, null, null, null);
```

Group by, having y order by

Recorrer el resultado

```
while (cu.moveToNext()) {  
    int Cod = cu.getInt(0);  
    String Nom = cu.getString(1);  
    ...  
}
```



ALMACENAMIENTO DE DATOS

- Bases de datos locales

- IMPORTANTE

- Las conexiones y los cursores consumen recursos.
 - Hay que cerrarlos cuando se termina de trabajar con ellos.

```
cu.close();  
bd.close();
```

- Para borrar una base de datos local:
 - Desde una actividad

```
this.deleteDatabase("NombreBD");
```



ALMACENAMIENTO DE DATOS

○ Ejercicio 3:

- Crear una aplicación donde se le pregunte al usuario una palabra.
 - Si existe en la base de datos se le avisa de ello
 - Si la palabra no existe se le pregunta si quiere añadirla la base de datos o no. Si responde afirmativamente, se añade a la base de datos.



ALMACENAMIENTO DE DATOS

○ Bases de datos locales

- Android incluye capa de abstracción mediante la biblioteca de persistencias **Room**
- Para usar Room en la app, agrega las siguientes dependencias en el fichero build.gradle de la app

```
dependencies {  
    def room_version = "2.6.1"  
  
    implementation("androidx.room:room-runtime:$room_version")  
  
    // If this project only uses Java source, use the Java annotationProcessor  
    // No additional plugins are necessary  
    annotationProcessor("androidx.room:room-compiler:$room_version")  
}
```



ALMACENAMIENTO DE DATOS

- Bases de datos locales
 - Estos son los tres componentes principales de Room
 - La **clase de la base de datos** que contiene la base de datos y sirve como punto de acceso principal para la conexión subyacente a los datos persistentes de la app
 - Las **entidades de datos** que representan tablas de la base de datos de tu app
 - Los **objetos de acceso a datos (DAOs)** que proporcionan métodos que tu app puede usar para consultar, actualizar, insertar y borrar datos en la base de datos



ALMACENAMIENTO DE DATOS

○ Bases de datos locales: **Entidades de datos**

- Se define una entidad de datos User. Cada instancia de User representa una fila en una tabla de user en la base de datos de la app.

Nombre de la tabla. Si no se añade, por defecto, el nombre de la clase.

```
@Entity(tableName = "users")
public class User {
    @PrimaryKey
    public int uid;

    @ColumnInfo(name = "first_name")
    public String firstName;

    @ColumnInfo(name = "last_name")
    public String lastName;
}
```

Nombre de la columna. Si no se añade la info, el nombre del campo.



ALMACENAMIENTO DE DATOS

○ Bases de datos locales: **Objeto de acceso a datos (DAO)**

- Se define un DAO llamado UserDao. UserDao proporciona los métodos que el resto de la app usa para interactuar con los datos de la tabla user.

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
            "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert
    void insertAll(User... users);

    @Delete
    void delete(User user);
}
```



ALMACENAMIENTO DE DATOS

○ Bases de datos locales: **base de datos**

- Se define una clase AppDatabase para contener la base de datos y definir su configuración. La clase de la base de datos debe cumplir con las siguientes condiciones:
 - La clase debe tener una anotación @Database que incluya un array entities que enumere todas las entidades de datos asociados con la base de datos.
 - Debe ser una clase abstracta que extienda RoomDatabase.
 - Para cada clase DAO que se asoció con la base de datos, esta base de datos debe definir un método abstracto que tenga cero argumentos y muestre una instancia de la clase DAO.

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```



ALMACENAMIENTO DE DATOS

- Bases de datos locales:

- Después de definir la entidad de datos, el DAO y el objeto de base de datos, se crea una instancia de la base de datos:

```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),  
    AppDatabase.class, "database-name").build();
```

- Room en principio no permite el acceso a la base de datos en el proceso principal para evitar que las consultas bloqueen la IU, . Se recomienda que las búsquedas DAO sean asíncronas. Para permitir la consulta en el proceso principal:

```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),  
    AppDatabase.class, "database-name").allowMainThreadQueries().build();
```

- Luego, se pueden usar los métodos abstractos de AppDatabase para obtener una instancia del DAO al igual que los métodos de la instancia del DAO para interactuar con la base de datos:

```
UserDao userDao = db.userDao();  
List<User> users = userDao.getAll();
```



ALMACENAMIENTO DE DATOS

◦ Ejercicio 4:

- Repetir el ejercicio 3 mediante Room. Crear una aplicación donde se le pregunte al usuario una palabra.
 - Si existe en la base de datos se le avisa de ello
 - Si la palabra no existe se le pregunta si quiere añadirla la base de datos o no. Si responde afirmativamente, se añade a la base de datos.

