Apartado 12:
Almacenamiento

Autores:
Víctor M. Rivas Santos / Miguel Á. García Cumbreras
(Antonio Rueda Ruiz)

Opciones para conseguir persistencia

- Preferencias compartidas
- Almacenamiento interno (en memoria)
- Almacenamiento externo (en tarjeta)
- Bases de datos SQLite
- A través de la red, en un servidor (no se ve en este apartado)

https://developer.android.com/guide/topics/data/data-storage.html

Almacenamiento de preferencias

- Almacenamiento de tipos de datos básicos como pares: *{llave,valor}*
- Persisten de una actividad a otra **e incluso de una sesión a otra**.
- La clase fundamental es *SharedPreferences* junto con *SharedPreferences.Editor*

Añadir, modificar, eliminar o recuperar preferencias

Modificar el valor de una preferencia

```
SharedPreferences preferencias = getSharedPreferences("nombre_fichero", 0);
SharedPreferences.Editor editorPreferencias = preferencias.edit();

editorPreferencias.putString("token_login", "Periquillo de los palotes"); // Añadir o modificar
editorPreferencias.remove("token_edad"); // Eliminar

editorPreferencias.commit();
```

Recuperar el valor de una preferencia

```
SharedPreferences preferencias = getSharedPreferences( "nombre_fichero", 0);

login=preferencias.getString("token_login", "VALOR_POR_DEFECTO");
```

El segundo parámetro (0) hace referencia al modo en que se abre: MODE_PRIVATE

Preferencias de la app

Modificar el valor de una preferencia

```
SharedPreferences preferencias = getDefaultSharedPreferences(getApplicationContext());

SharedPreferences.Editor editorPreferencias = preferencias.edit();

editorPreferencias.putString("token_login", "Periquillo de los palotes"); // Añadir o modificar
editorPreferencias.remove("token_edad"); // Eliminar

editorPreferencias.commit();
```

Recuperar el valor de una preferencia

```
SharedPreferences preferencias = getDefaultSharedPreferences(getApplicationContext());

login=preferencias.getString("token_login", "VALOR_POR_DEFECTO");
```

Preferencias de la activity

Modificar el valor de una preferencia

```
SharedPreferences preferencias = getPreferences(0);

SharedPreferences.Editor editorPreferencias = preferencias.edit();

editorPreferencias.putString("token_login", "Periquillo de los palotes"); // Añadir o modificar
editorPreferencias.remove("token_edad"); // Eliminar

editorPreferencias.commit();
```

Recuperar el valor de una preferencia

```
SharedPreferences preferencias = getPreferences(0);

login=preferencias.getString("token_login", "VALOR_POR_DEFECTO");
```

Almacenamiento interno (a)

Ejemplo: Almacenamiento_Ficheros

- Los archivos de almacenamiento interno son privados para la aplicación
  - Desaparecen al desinstalarla
- Primera opción: ficheros de texto raw (directorio *res/raw*), solo lectura

```
private void leerFicheroRaw() {
 String mensaje="";
 try {
  InputStream ficheroRaw = getResources().openRawResource(R.raw.test_raw);
  BufferedReader buffer = new BufferedReader(new InputStreamReader(ficheroRaw));
  String linea="";
  int numLinea=0;
  while( buffer.ready() ) {
   linea = buffer.readLine();
   mensaje += "#"+(++numLinea)+": " + linea + "\n";
  }
  ficheroRaw.close();
  ((TextView) findViewById(R.id.tvHW)).setText(mensaje);
 } catch (Exception ex) {
  Toast.makeText(this, "leerFicheroRaw: "+"Error: "+ ex.getMessage(), Toast.LENGTH_SHORT).show();
 }
}
```

Almacenamiento interno (b)

- Segunda opción: ficheros de lectura y escritura
- Escritura:

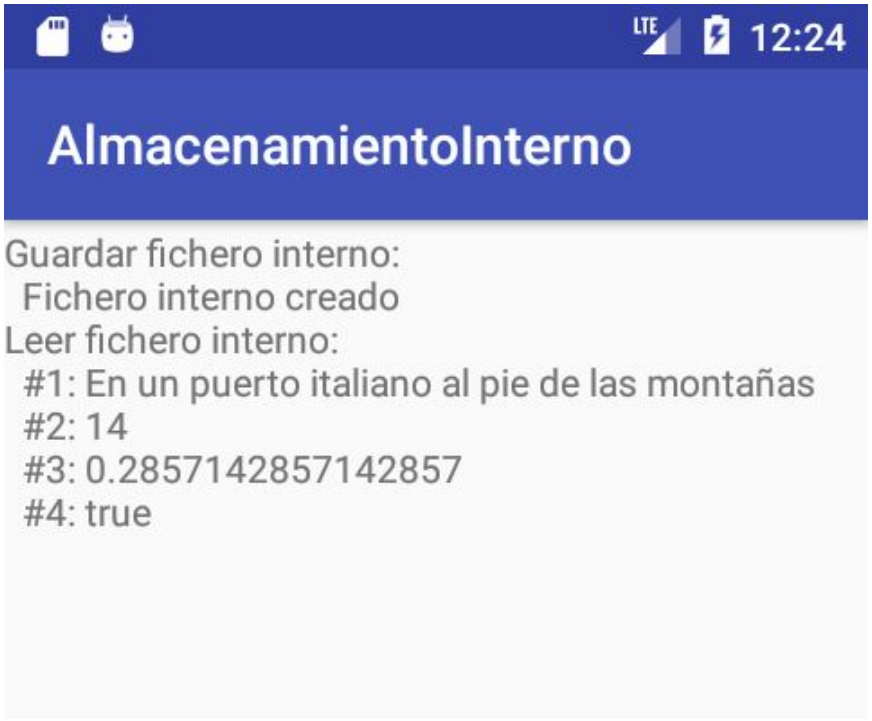
```
private void guardarFicheroInterno() {
 try {
  String mensaje = "Guardar fichero interno: \n";
  OutputStreamWriter ficheroInterno = new OutputStreamWriter(openFileOutput("fichero_interno.txt", Context.MODE_PRIVATE));
  ficheroInterno.write("En un puerto italiano al pie de las montañas\n");
  ficheroInterno.write((2 * 7)+"\n");
  ficheroInterno.write((2.0 / 7)+"\n");
  ficheroInterno.write((1==1)+"\n");
  ficheroInterno.close();
  mensaje+=" Fichero interno creado.\n";
  ((TextView) findViewById(R.id.tvHW)).setText(mensaje);
 } catch (Exception ex) {
  Toast.makeText(this, "guardarFicheroInterno: " + "Error: " + ex.getMessage(), Toast.LENGTH_SHORT).show();
 }
}
```

Almacenamiento interno (c)

- Segunda opción: ficheros de lectura y escritura
  - Lectura:

```
private void leerFicheroInterno() {
 String mensaje = ((TextView) findViewById(R.id.tvHW)).getText()
   +"\n"
   +"Leer fichero interno: \n";
 try {
  InputStreamReader ficheroInterno=new InputStreamReader(openFileInput("fichero_interno.txt"));
  BufferedReader buffer = new BufferedReader(ficheroInterno);
  String linea = "";
  int numLinea = 0;
  while (buffer.ready()) {
   linea = buffer.readLine();
   mensaje += " #" + (++numLinea) + ": " + linea + "\n";
  }
  ficheroInterno.close();
  ((TextView) findViewById(R.id.tvHW)).setText(mensaje);
 } catch (Exception ex) {
  Toast.makeText(this, "guardarFicheroInterno: " + "Error: " + ex.getMessage(), Toast.LENGTH_SHORT).show();
 }
}
```

Almacenamiento interno (d)



Almacenamiento externo

- Se realiza normalmente en la tarjeta SD extraíble…
- ...aunque también puede ser en un dispositivo interno.
- Es un espacio compartido por todas las aplicaciones
- Existen algunos directorios estándar como:
  - Environment.DIRECTORY_MUSIC
  - Environment.DIRECTORY_PICTURES
  - Environment.DIRECTORY_RINGTONES

Antes de usar el almacenamiento externo

- Solicitar permisos al SO en el fichero manifest:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

- Comprobar que el dispositivo está montado y si permite escribir o solo leer:

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
 String state = Environment.getExternalStorageState();
 return (Environment.MEDIA_MOUNTED.equals(state));
}


/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
 String state = Environment.getExternalStorageState();
 return (Environment.MEDIA_MOUNTED.equals(state) ||
  Environment.MEDIA_MOUNTED_READ_ONLY.equals(state));
}
```

Almacenamiento externo: ficheros compartidos

- Usamos *getExternalStoragePublicDirectory()* para crear/acceder ficheros y directorios en ubicaciones accesibles para cualquier app

```
public File crearDirectorioImagenes() {
 String nombreDirectorio = "mis_foticos";
 File file = new File(Environment.getExternalStoragePublicDirectory(
     Environment.DIRECTORY_PICTURES), nombreDirectorio);
 if (file.exists()) {
  Toast.makeText(this, "crearDirectorioImagenes: Ya existía el directorio "
            + nombreDirectorio
            , Toast.LENGTH_SHORT).show();
 } else {
  if (!file.mkdirs()) {
    Toast.makeText(this, "crearDirectorioImagenes: No se pudo crear el directorio " + nombreDirectorio,
Toast.LENGTH_SHORT).show();
  } else {
    String mensaje = ((TextView) findViewById(R.id.tvHW)).getText()
        + "\n"
        + "crearDirectorioImagenes: \n";
    mensaje += " Directorio " + nombreDirectorio + " creado.\n";
    ((TextView) findViewById(R.id.tvHW)).setText(mensaje);
  }
 }
 return file;
}
```

Almacenamiento externo: ficheros privados

- Solo son modificables por nuestra app, pero pueden ser leídos por otras apps.
- Al desinstalar la aplicación se eliminan
- Obtenemos el directorio RAíZ para los ficheros de nuestra aplicación con *getExternalFilesDir()*.

Escritura de ficheros externos privados

```
private void guardarFicheroExternoPrivado() {
  String mensaje = "";
  if( !isExternalStorageWritable()) {
    Toast.makeText(this, "guardarFicheroExternoPrivado: "
            + "Error: No se puede escribir en dispositivo externo"
            , Toast.LENGTH_LONG).show();
    return;
  }
  try {
    File fichero = new File(getExternalFilesDir(null), "fichero_externo.txt");
    OutputStreamWriter ficheroExterno = new OutputStreamWriter(new FileOutputStream(fichero));
    ficheroExterno.write("En un país multicolor\n");
    ficheroExterno.write((2 - 3) + "\n");
    ficheroExterno.write((2.0 / 3) + "\n");
    ficheroExterno.write((1 == 0) + "\n");
    ficheroExterno.close();
    mensaje += " Fichero externo creado.\n";
    ((TextView) findViewById(R.id.tvHW)).setText(mensaje);
  } catch (Exception ex) {
    Toast.makeText(this, "guardarFicheroExternoPrivado: "
            + "Error: "
            + ex.getMessage(), Toast.LENGTH_LONG).show();
  }
}
```

Lectura de ficheros externos privados

```
private void leerFicheroExternoPrivado() {
  String mensaje = "Leer fichero externo privado: \n";
  if( !isExternalStorageReadable()) {
    Toast.makeText(this, "leerFicheroExternoPrivado: "
            + "Error: No se puede leer de dispositivo externo"
            , Toast.LENGTH_LONG).show();
    return;
  }
  try {
    File fichero = new File(getExternalFilesDir(null), "fichero_externo.txt");
    InputStreamReader ficheroInterno = new InputStreamReader(new FileInputStream(fichero));
    BufferedReader buffer = new BufferedReader(ficheroInterno);
    String linea = "";
    int numLinea = 0;
    while (buffer.ready()) {
      linea = buffer.readLine();
      mensaje += " #" + (++numLinea) + ": " + linea + "\n";
    }
    ficheroInterno.close();
    ((TextView) findViewById(R.id.tvHW)).setText(mensaje);
  } catch (Exception ex) {
    Toast.makeText(this, "leerFicheroExternoPrivado: " + "Error: " + ex.getMessage()
            , Toast.LENGTH_LONG).show();
  }
}
```

Procesamiento de JSON

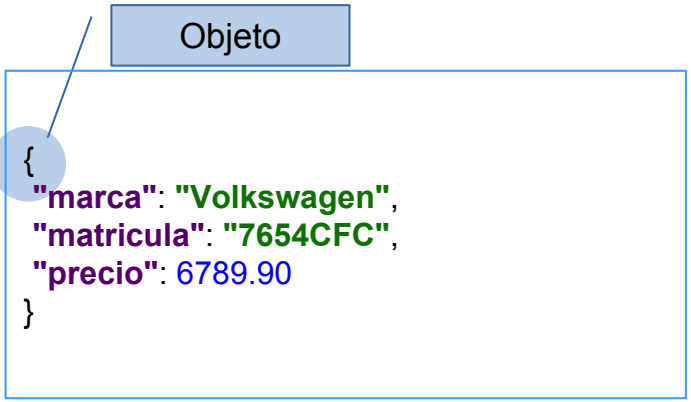
Ejemplo: FicheroJSON

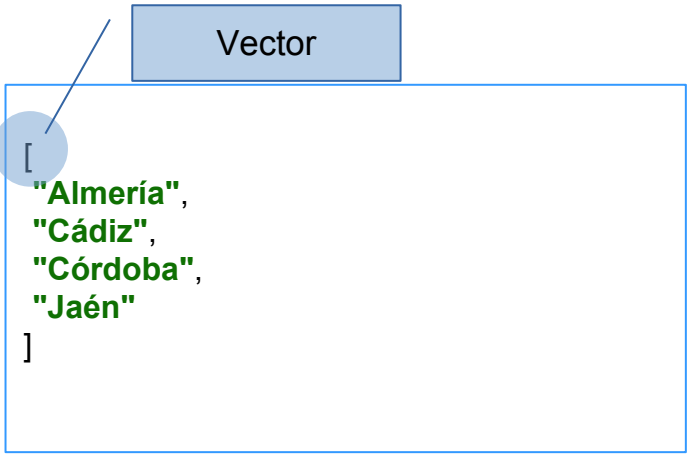
- Como es conocido, JSON es un formato estándar de representación de objetos y vectores.
- Podemos utilizarlo tanto para almacenar/recuperar información en ficheros…
- … como para enviar/recibir información por la red.

Imprescindible para JSON

- Las clases *JSONObject* y *JSONArray* son la base de todo el procesamiento/generación de JSON
- Es IMPRESINCIDIBLE conocer el formato de los datos que vamos a leer/generar

Ejemplos simples de JSON





Procesamiento de objetos simples

```
public void procesarObjetoSimple() {
  String cadena = "{ \"marca\": \"Volkswagen\"" +
      ", \"matricula\": \"7654CFC\"" +
      ", \"precio\": 6789.90" +
      "}";


  try {
    JSONObject object = (JSONObject) new JSONObject(cadena);


    String marca = object.getString("marca");
    String matricula = object.getString("matricula");
    double precio = object.getDouble("precio");


    tvMje.setText(
        "procesarObjeto: \n"
          + " - Marca: " + marca + "\n"
          + " - Matrícula: " + matricula + "\n"
          + " - Precio: " + precio + "\n"
    );
  } catch (JSONException ex) {
    Toast.makeText(this, "procesarObjetoSimple: " + ex.getMessage()
                                          , Toast.LENGTH_SHORT).show();
  }
}
```

Procesamiento de vectores simples

```
public void procesarVectorSimple() {
  String ciudades="[\"Almería\", \"Cádiz\", \"Córdoba\", \"Jaén\"]";
  String msj = "Procesar Vector Simple: \n" +
      " - Ciudades: \n";
  try {
    JSONArray vector = (JSONArray) new JSONArray(ciudades);
    for (int i = 0; i < vector.length(); ++i) {
      msj += "  - "+vector.getString(i)+"\n"; // usa posición (i)

    }
    tvMje.setText(msj);
  } catch (JSONException ex) {
    Toast.makeText(this, "procesarVectorSimple: " + ex.getMessage()
            , Toast.LENGTH_LONG).show();
  }
}
```

Ejemplos complejos de JSON

Vector

```
[
 {
  "marca": "Volkswagen",
  "matricula": "7654CFC",
  "precio": 6789.9,
  "itvs": [
   2005,
   2007,
   2009,
   2010,
   2011,
   2013,
   2014,
   2015,
   2016,
   2017
  ]
 },
 {
  "marca": "Seat",
  "matricula": "8762JRV",
  "precio": 16353.1,
  "itvs": []
 }
]
```

Objeto

```
{
 "titulo": "Pedro y el lobo",
 "autor": "Sergei Prokofiev",
 "protagonistas": [ {
  "nombre": "Pedro",
  "instrumentos": ["violín", "violas"
        , "violonchelos", "contrabajo"]
 },
 {
  "nombre": "Abuelo",
  "instrumentos": ["fagot"]
 },
 {
  "nombre": "Pájaro",
  "instrumentos": ["flauta travesera"]
 },
 {
  "nombre": "Pato",
  "instrumentos": ["oboe"]
 },
 {
  "nombre": "Gato",
  "instrumentos": ["clarinete"]
 },
 {
  "nombre": "Lobo",
  "instrumentos": ["trompa"]
 },
 {
  "nombre": "Cazadores",
  "instrumentos": ["timables","bombo"]
 }
 ]
}
```

Procesamiento de objetos complejos

```
public void procesarObjetoComplejo() {
  String mje = "";
  try {
    JSONObject object = (JSONObject) new JSONObject(musica);
    String titulo = object.getString("titulo");
    String autor = object.getString("autor");
    JSONArray protagonistas = object.getJSONArray("protagonistas");


    mje += "Procesar Objeto Complejo:\n"
        + " - Titulo: " + titulo + "\n"
        + " - Autor: " + autor + "\n"
        + " - Protagonistas: " + protagonistas.length() + "\n";


    for (int j = 0; j < protagonistas.length(); ++j) {
      JSONObject protagonista = protagonistas.getJSONObject(j);
      mje += "  - Nombre: " + protagonista.getString("nombre") + "\n";
      mje += "  - Instrumentos: \n" ;
      JSONArray instrumentos=protagonista.getJSONArray("instrumentos");
      for( int k=0; k<instrumentos.length(); ++k ) {
        mje+="   - "+instrumentos.getString(k)+", ";
      }
      mje+="\n";
    }
    tvMje.setText(mje);
  } catch (JSONException ex) {
    Toast.makeText(this, "procesarObjetoComplejo: " + ex.getMessage()
            , Toast.LENGTH_LONG).show();
  }
}
```

Procesamiento de vectores complejos

```
public void procesarVectorComplejo() {
  String msj = "";
  try {
    JSONArray vector = (JSONArray) new JSONArray(vehiculos);
    for (int i = 0; i < vector.length(); ++i) {
      JSONObject vehiculo = vector.getJSONObject(i);
      msj += "procesarVectorComplejo - vehiculo [" + i + "]: \n"
          + " - Marca: " + vehiculo.getString("marca") + "\n"
          + " - Matrícula: " + vehiculo.getString("matricula") + "\n"
          + " - Precio: " + vehiculo.getDouble("precio") + "\n"
          + " - Núm. ITVs: " + vehiculo.getJSONArray("itvs").length() + "\n";


      for (int j = 0; j < vehiculo.getJSONArray("itvs").length(); ++j) {
        msj += "  - ITVs[" + j + "]: " + vehiculo.getJSONArray("itvs").getInt(j) + "\n";
      }
    }
    tvMje.setText(msj);
  } catch (JSONException ex) {
    Toast.makeText(this, "procesarObjeto: " + ex.getMessage()
                                          , Toast.LENGTH_LONG).show();
  }
}
```

Generar JSON para objetos y vectores (a)

Objeto Simple

```
class Vehiculo {
    String marca = "";
    String matricula = "";
    double precio = 0;


    public JSONObject toJSON() {
       JSONObject jsonObject = new JSONObject();
       try {
          jsonObject.put("marca", marca);
          jsonObject.put("matricula", matricula);
          jsonObject.put("precio", precio);
          return jsonObject;
       } catch (JSONException e) {
          e.printStackTrace();
          return null;
       }
    }
}

// Para usarlo
Vehiculo taxi = new Vehiculo("Toyota", "8745JVC" , 22000.25);
String vehiculoJSON = taxi.toJSON().toString();
```

Generar JSON para objetos y vectores (b)

Vector Simple

```
List<String> nombres = new ArrayList<>();
nombres.add("Pepe");
nombres.add("Luis");
nombres.add("Angel");


JSONArray vector = new JSONArray(nombres);


String nombresJSON = vector.toString();
```

Generar JSON para objetos y vectores (c)

Objeto complejo

```
class Receta {
   String nombre;
   int tiempo;
   List<Ingrediente> ingredientes;

   public JSONObject toJSON() {
      try {
         JSONObject laReceta = new JSONObject();
         laReceta.put("nombre", nombre);
         laReceta.put("tiempo", tiempo);
         JSONArray losIngredientes = new JSONArray();
         for (int i = 0; i < ingredientes.size(); ++i) {
            losIngredientes.put(ingredientes.get(i).toJSON());
         }
         laReceta.put("ingredientes", losIngredientes);
         return laReceta;
      } catch (JSONException e) {
         e.printStackTrace();
         return null;
      }
   }
}
```

Generar JSON para objetos y vectores (d)

Vector complejo

```
List<Receta> menuDelDia = new ArrayList<>();
List<Ingrediente> ingredientes = new ArrayList<>();
ingredientes.add(new Ingrediente("Pollo", 3.00));
ingredientes.add(new Ingrediente("Vino", 0.25));
ingredientes.add(new Ingrediente("Cebolla", 0.300));
ingredientes.add(new Ingrediente("Pastilla Avecrem", 1));
menuDelDia.add(new Receta("Pollo al horno", 90, ingredientes));


ingredientes = new ArrayList<>();
ingredientes.add(new Ingrediente("Lechuga", 0.2));
ingredientes.add(new Ingrediente("Tomate", 0.3));
ingredientes.add(new Ingrediente("Pepino", 0.15));
menuDelDia.add(new Receta("Ensalada", 15, ingredientes));



JSONArray recetario = new JSONArray();
for( int i=0; i<menuDelDia.size(); ++i ) {
    recetario.put(menuDelDia.get(i).toJSON());
}
String recetarioJSON = recetario.toString();
```

Guardar JSON en fichero

```
File fichero = new File(getExternalFilesDir(null), "fichero_externo.json");

OutputStreamWriter ficheroExterno = new OutputStreamWriter(new FileOutputStream(fichero));

ficheroExterno.write(crearJSONVectorComplejo());

ficheroExterno.close();
```

Leer JSON desde fichero

```
File fichero = new File(getExternalFilesDir(null), "fichero_externo.json");
InputStreamReader ficheroExterno = new InputStreamReader(new FileInputStream(fichero));
JsonReader jsonReader = new JsonReader(ficheroExterno);
jsonReader.beginArray(); // Comienza a leer el vector de recetas
while (jsonReader.hasNext()) {
  jsonReader.beginObject(); // Comienza a leer el una receta
  while (jsonReader.hasNext()) {
     String token = jsonReader.nextName();
     if (token.equals("nombre")) { nombreReceta = jsonReader.nextString();}
     else if (token.equals("tiempo")) { tiempo = jsonReader.nextInt();}
     else if (token.equals("ingredientes")) {
        ingredientes = new ArrayList<>();
        jsonReader.beginArray();// Comienza a leer ingredientes para esta receta
        while (jsonReader.hasNext()) {
          jsonReader.beginObject();
          while (jsonReader.hasNext()) {
            // Comienza a leer un ingrediente
            token = jsonReader.nextName();
           if (token.equals("nombre")) { nombreIngrediente = jsonReader.nextString(); }
           else if (token.equals("cantidad")) { cantidad = jsonReader.nextDouble();}
        }
        jsonReader.endObject();
        ingredientes.add(new Ingrediente(nombreIngrediente, cantidad));
       }
       jsonReader.endArray();
    } else {
       jsonReader.skipValue();
    }
  }
  jsonReader.endObject();
  menuDelDia.add(new Receta(nombreReceta, tiempo, ingredientes));
}
jsonReader.endArray();
ficheroExterno.close();
```

Destacar:

- JsonReader
- beginArray()
- hasNext()
- nextName
- nextString
  nextInt …
- endObject
- endArray
- skipValue
- peek

SQLite

Ejemplo: Almacenamiento_SQLite

- Nos proporciona acceso a BBDD con una funcionalidad básica, pero suficiente
  - Tipos de datos: *null, integer, real, text, blob*
- Utilizaremos dos clases principalmente:
  - La interfaz *SQLiteOpenHelper*
  - La clase *SQLiteDatabase*

https://developer.android.com/reference/android/database/sqlite/package-summary.html

La interfaz *SQLiteOpenHelper*

- Facilita (por eso lo de *helper*) la automatización de ciertas tareas; fundamentalmente:
  - Crear la BBDD
  - Alterar la BBDD (es decir, su “estructura”)

SQLiteHelper del ejemplo

```
public class DBMS extends SQLiteOpenHelper {

   public static final String DATABASE_NAME = "alojamiento_vacaciones.db";
   public static final int DATABASE_VERSION = 1;

   DBMS(Context context) {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }

   @Override
   public void onCreate(SQLiteDatabase database) {
      database.execSQL("create table hoteles (id integer primary key autoincrement" +
                  ", nombre text, estrellas integer, localidad text)");

      database.execSQL("create table vacaciones(id integer primary key autoincrement" +
            ", fecha_inicio text, fecha_fin text" +
            ", id_hotel integer references hoteles(id)", valoracion integer)");
   }

   @Override
   public void onUpgrade(SQLiteDatabase database, int old_version, int new_version) {
      database.execSQL("drop table if exists hoteles");
      database.execSQL("drop table if exists vacaciones");
      onCreate(database);
   }

   public void recrea(SQLiteDatabase database) {
      database.execSQL("drop table if exists hoteles");
      database.execSQL("drop table if exists vacaciones");
      onCreate(database);
   }
}
```

Operaciones sobre la bbdd (a)

1. Crear/abrir la BBDD

```
dbms = new DBMS(this);
```

2. Insertar registro con *insert* (opción recomendada)

```
/**
 * Añade un nuevo hotel usando el método insert
 *
 * @param nombre
 * @param estrellas
 * @param localidad
 * @return Id asignado a dicho hotel
 */
public long añadirHotel_insert(String nombre, int estrellas, String localidad) {
    SQLiteDatabase sqlDB = dbms.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("nombre", nombre);
    values.put("estrellas", estrellas);
    values.put("localidad", localidad);
    long nuevoId = sqlDB.insert("hoteles", null, values);
    sqlDB.close();
    return nuevoId;
}
```

Operaciones sobre la bbdd (b)

3. Insertar registro con *execSQL*

```
/**
 * Añade un nuevo hotel pero usando el método execSQL
 *
 * @param nombre
 * @param estrellas
 * @param localidad
 */
public void añadirHotel_execSQL(String nombre, int estrellas, String localidad) {
    SQLiteDatabase sqlDB = dbms.getWritableDatabase();

    sqlDB.execSQL("insert into hoteles values(null, ?, ?, ?);"

                , new String[]{nombre, "" + estrellas, localidad});

    sqlDB.close();
}
```

Operaciones sobre la bbdd (c)

4. Modificar un registro con *update* (opción recomendada)

```
/**
 * Modifica los datos de un hotel usando el método UPDATE
 *
 * @param id       Id del hotel a modificar
 * @param nombre
 * @param estrellas
 * @param localidad
 */
public void modificarHotel_update(long id, String nombre, int estrellas, String localidad) {
    SQLiteDatabase sqlDB = dbms.getWritableDatabase();
    ContentValues values = new ContentValues();


    values.put("nombre", nombre);
    values.put("estrellas", estrellas);
    values.put("localidad", localidad);


    sqlDB.update("hoteles", values, "id=?", new String[]{"" + id});
    sqlDB.close();
}
```

Operaciones sobre la bbdd (d)

4. Modificar un registro con *execSQL*

```
/**
 * Modifica los datos de un hotel usando el método execSQL
 *
 * @param id       Id del hotel a modificar
 * @param nombre
 * @param estrellas
 * @param localidad
 */
public void modificarHotel_execSQL(long id, String nombre, int estrellas, String localidad) {

    SQLiteDatabase sqlDB = dbms.getWritableDatabase();

    sqlDB.execSQL("update hoteles set nombre=?, estrellas=?, localidad=? where id=?"

                , new String[]{nombre, "" + estrellas, localidad, "" + id});

    sqlDB.close();
}
```

Operaciones sobre la bbdd (e)

5. Consultas con *query* (opción preferida)

```
/**
 * Imprime una lista de los hoteles haciendo la consulta con QUERY
 *
 * @param limite Máximo número de hoteles a recuperar
 */
public void listarHoteles_query(int limite) {
   SQLiteDatabase sqlDB = dbms.getReadableDatabase();
   Cursor cursor = sqlDB.query(false, "hoteles"
         , new String[]{"*"}, null, null, null, null,null, "" + limite);


   if (cursor == null) {
      nuevoMensaje("  - NO HAY HOTELES PARA MOSTRAR", false);
      return;
   }
   cursor.moveToFirst();
   while (!cursor.isAfterLast()) {
      nuevoMensaje("  - Hotel: "
                  + cursor.getInt(0)
                  + ": "
                  + cursor.getString(1)
                  + " "
                  + pintaEstrellas(cursor.getInt(2))
            , false);
      cursor.moveToNext();
   }
   cursor.close();
}
```

Ejemplos *query* más complejos

```
Cursor cursor = sqlDB.query(
        false // Distinct
        , "hoteles" // Tabla
        , new String[]{"id", "nombre", "estrellas", "length(nombre)"} // Columnas
        , "id=?" // Cláusula where
        , new String[]{"4"} // Vector de argumentos
        , "estrellas" // Cláusula group by.
        , "length(nombre)<10" // Cláusula having
                                                // (solo sirve cuando también hay cláusula group by)
        , "estrellas DESC, nombre" // Cláusula order by.
        , "10" // Cláusula limit
);
```

```
Cursor cursor = sqlDB.query(
        false // Distinct
        , "vacaciones, hoteles" // Dos tablas
        , new String[]{"vacaciones.id", "fecha_inicio", "fecha_fin"
                , "valoracion", "hoteles.nombre"}
        , "hoteles.id=vacaciones.id_hotel" // where
        , null // Vector de argumentos
        , null // Cláusula group by.
        , null // Cláusula having
        , null // Cláusula order by.
        , "10" // Cláusula limit
);
```

Operaciones sobre la bbdd (f)

6. Consultas con rawQuery

```
 * Imprime una lista de los hoteles haciendo la consulta con RAWQUERY
 *
 * @param limite Máximo número de hoteles a recuperar
 */
public void listarHoteles_rawQuery(int limite) {
    Cursor cursor = dbms.getReadableDatabase().
            rawQuery("select * from hoteles limit ?", new String[]{"" + limite});
    if (cursor == null) {
        nuevoMensaje("  - NO HAY HOTELES PARA MOSTRAR", false);
        return;
    }
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        nuevoMensaje("  - Hotel: "
                    + cursor.getInt(0)
                    + ": "
                    + cursor.getString(1)
                    + " ("+ pintaEstrellas(cursor.getInt(2))+")"
                , false);
        cursor.moveToNext();
    }
    cursor.close();
}
```

Otra forma de recorrer el cursor

```
if (cursor == null) {
    nuevoMensaje("  - NO HAY VACACIONES PARA MOSTRAR", false);
    return;
}
cursor.moveToFirst();
do {
    nuevoMensaje("  - Vacaciones: "
                + cursor.getInt(0)
                + ": Desde "
                + cursor.getString(1)
                + "\n        hasta "
                + cursor.getString(2)
                + "\n        En "
                + cursor.getString(4)
                + " ("+ pintaValoracion(cursor.getInt(3))+")"
            , false);
} while (cursor.moveToNext());
cursor.close();
```

Operaciones sobre la bbdd (g)

7. Eliminación de registros con *delete*

```
/**
 * Elimina todos los registros de una tabla
 *
 * @param tabla Nombre de la tabla a eliminar
 */
public void borrarTabla(String tabla) {
    SQLiteDatabase sqlDB = dbms.getWritableDatabase();
    sqlDB.delete(tabla, null, null);
    sqlDB.close();
}



/**
 * Eliminaun hotel
 *
 * @param id Identificador del hotel
 */
public void eliminarHotel_delete(long id) {
    SQLiteDatabase sqlDB = dbms.getWritableDatabase();
    sqlDB.delete("hoteles", "id=?", new String[] {""+id});
    sqlDB.close();
}
```

Consideraciones finales

- Es preferible usar los métodos *insert*, *update, delete, query* y *rawQuery* para las tareas que están pensados.
  - Nos proporcionan información sobre el número de filas afectadas, último ID generado, etc.
- No obstante, usaremos *execSQL* cuando queramos crear o alterar la estructura de las tablas de la BBDD.
  - *execSQL* devuelve void.