SESIÓN 9

- Directorios *assets* y *res/raw*
- Analizar JSON con JSONObject
- Analizar XML con SAX
- *Almacenamiento persistente*

Ficheros de la aplicación

- Si una aplicación necesita usar ficheros estáticos (audio, video, texto …), estos deben existir en el directorio *res/raw* o en el directorio *assets*.
- Características comunes de los directorios *assets* y *res/raw*:
  - Sus ficheros no se pueden modificar en tiempo de ejecución. Se empaquetan junto con la aplicación
  - Para poder usar estos directorios, es necesario crearlos previamente en Android Studio
  - Al abrir un fichero, generan un InputStream.

```
// Asset:
AssetManager am = getAssets();
InputStream f = am.open("directorio/fichero");
// res/raw:
InputStream f = getResources().openRawResource(R.raw.test);
```

https://developer.android.com/reference/android/content/res/AssetManager

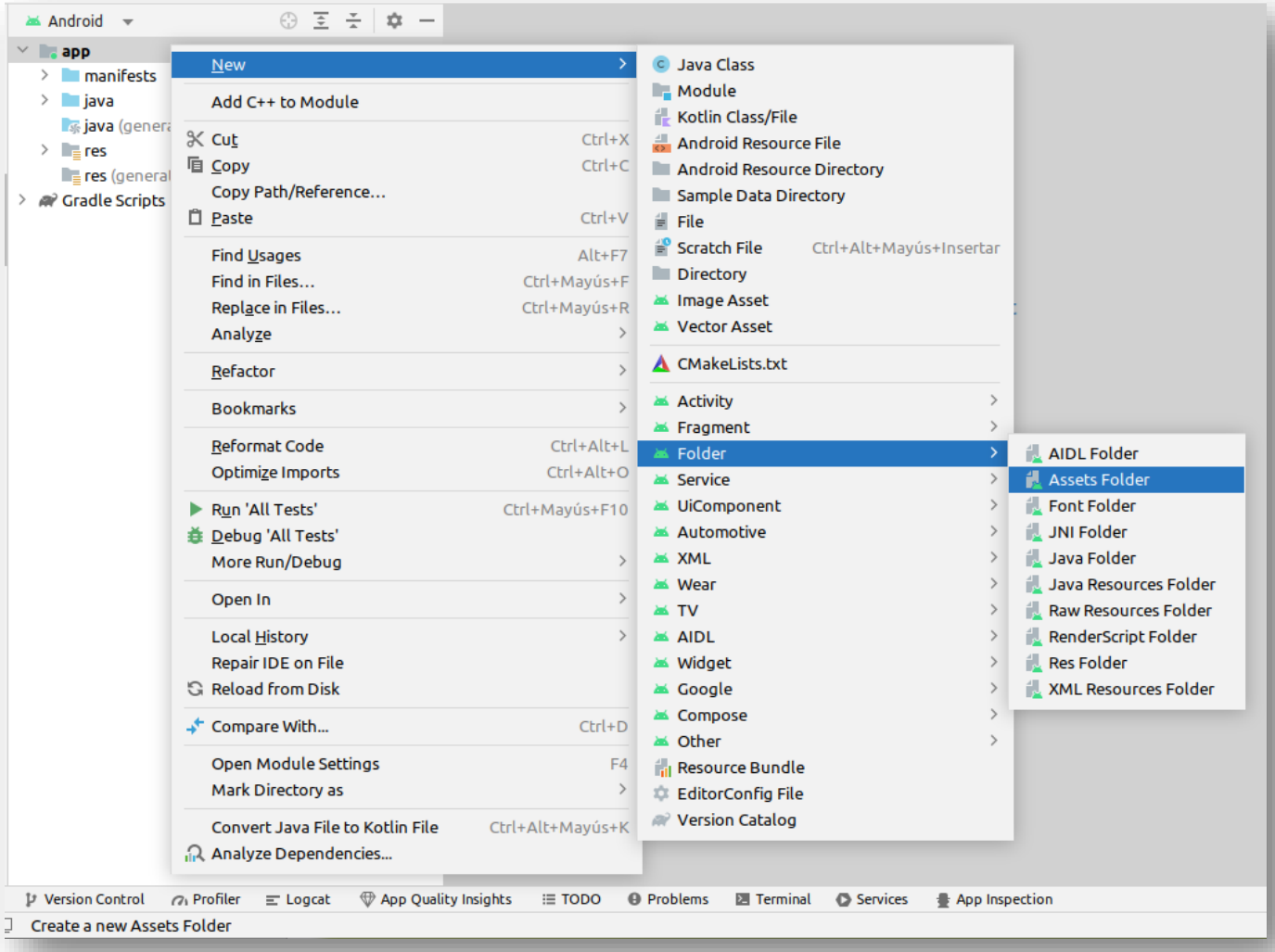
Ficheros de la aplicación

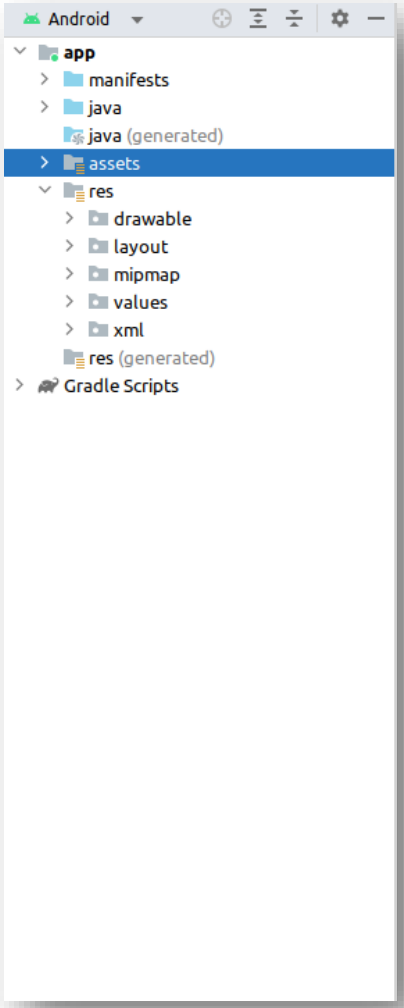
Diferencias de *assets* y *res/raw*:

- *asset* puede usar una estructura de subdirectorios
- *asset* es más flexible en los nombres de ficheros (puede tener mayúsculas y espacios)
- *asset* permite listar los ficheros en tiempo de ejecución
- *asset* detecta errores de apertura en tiempo de ejecución, mientras que *res/raw* en compilación
- Un fichero en *res/raw* es referenciado por su identificador de recurso (`R.raw.nombre`) mientras que un fichero de *asset* debe identificarse por su nombre
- El acceso a ficheros *asset* es más lento que a ficheros en *res/raw*

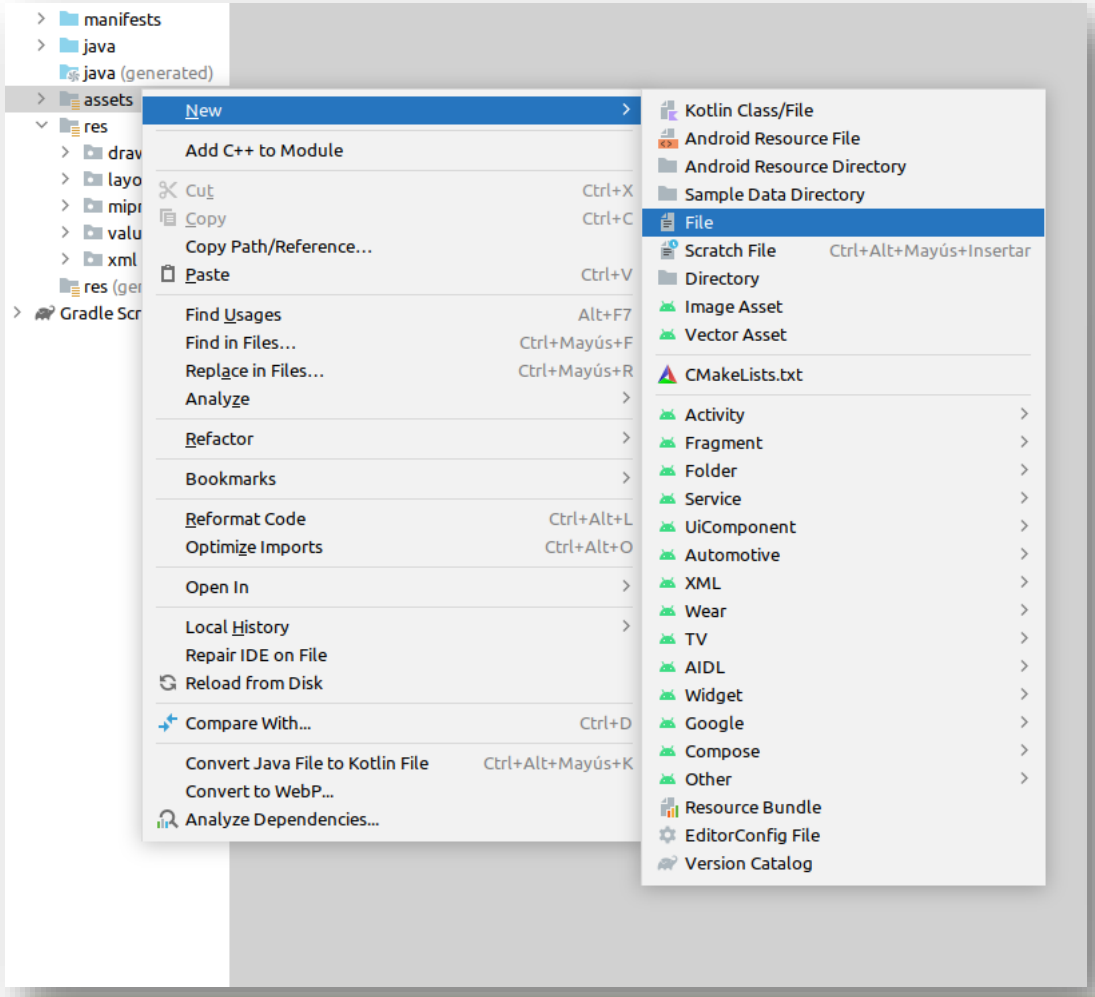
https://medium.com/mobile-app-development-publication/assets-or-resource-raw-folder-of-android-5bdc042570e0

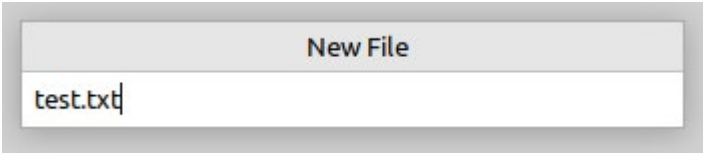
Directorio *assets*: creación

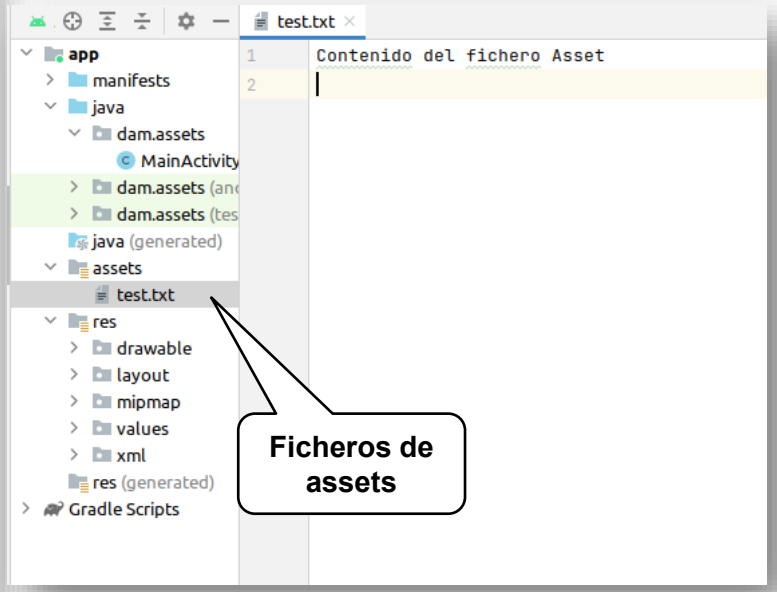




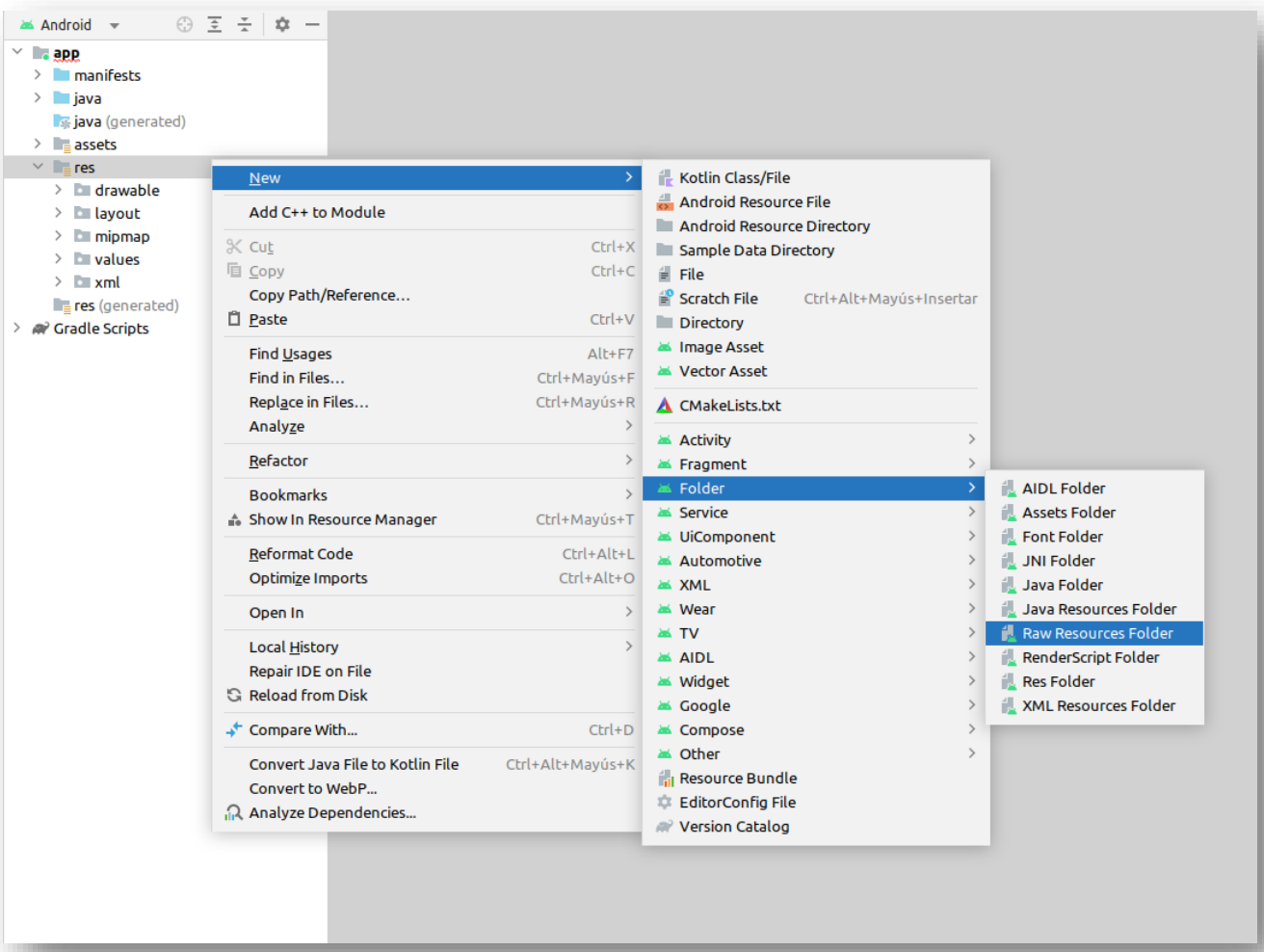
Directorio *assets*: creación de un fichero

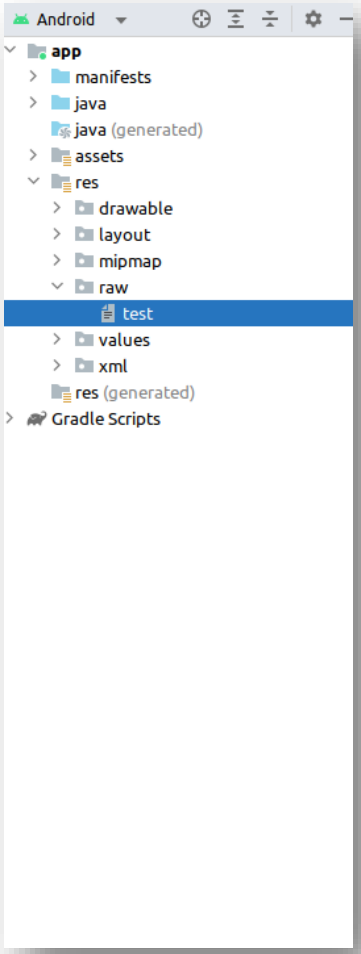






Directorio *res/raw*: creación





Directorio *assets*: manejo

```
final StringBuilder respuesta= new StringBuilder();
String linea;

// Leer fichero Asset
final AssetManager am = getAssets();
try {
   final is inputStream = am.open("test.txt");
   final BufferedReader bufferedReader =
         new BufferedReader(new InputStreamReader(is));
   while ((linea = bufferedReader.readLine()) != null) {
      respuesta.append(linea).append("\n");
   }
   inputStream.close();
} catch (IOException e) {
   e.printStackTrace();
   Log.d ("Mensaje","Error al recuperar el fichero “
         +e.toString());
}
```

```
// Leer fichero res/raw

try {
   InputStream is = getResources().openRawResource(R.raw.test);
   final BufferedReader bufferedReader =
         new BufferedReader(new InputStreamReader(is));
   while ((linea = bufferedReader.readLine()) != null) {
      respuesta.append(linea).append("\n");
   }
   inputStream.close();
} catch (IOException e) {
   e.printStackTrace();
   Log.d ("Mensaje","Error al recuperar el fichero "
         +e.toString());
}
```

Analizador JSON

La clase `JSONObject` permite analizar documentos JSON

Método	Descripción
`new JSONObject(String json)`	Crea un nuevo objeto JSON a partir de un String con contenido JSON
`getString(String key)`	Obtiene una cadena (String) asociada a la clave dada
`getInt(String key)`	Obtiene un entero (int) asociado a la clave dada
`getJSONObject(String key)`	Devuelve otro objeto JSONObject anidado bajo la clave especificada
`getJSONArray(String key)`	Devuelve un array JSONArray bajo la clave especificada
`getString(int index)`	(de JSONArray) obtiene una cadena desde una posición del array
`getJSONObject(int index)`	(de JSONArray) obtiene un JSONObject en la posición indicada del array

Ejemplo

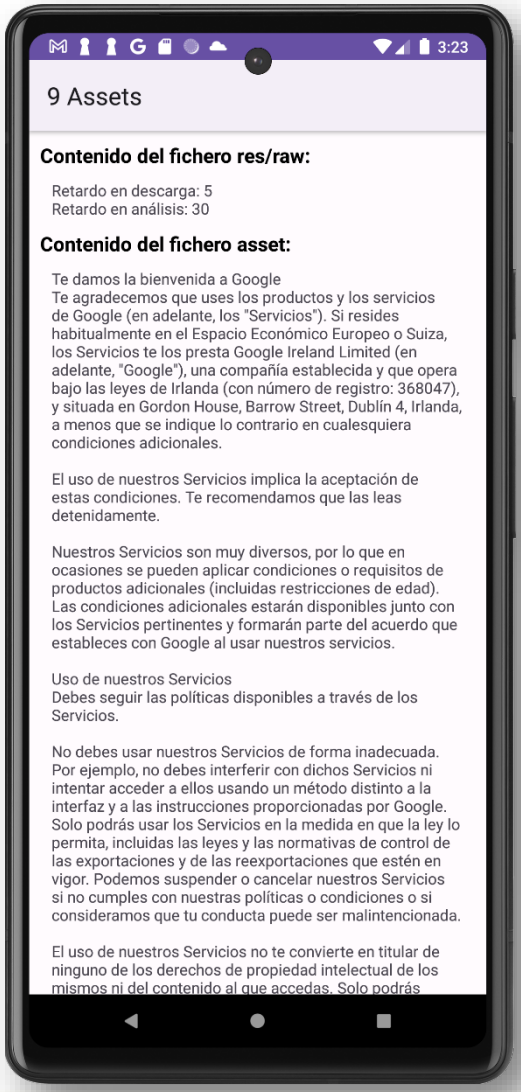
```
{
  "nombre": "Laura",
  "edad": 28,
  "direccion": {
    "ciudad": "Madrid",
    "codigoPostal": "28001"
  },
  "hobbies": [
    "leer",
    "senderismo",
    {
      "nombre": "fotografía",
      "nivel": "intermedio"
    }
  ]
}
```

```
try {
    JSONObject jsonObject = new JSONObject(jsonStr);
    // Propiedades simples
    String nombre = jsonObject.getString("nombre");
    int edad = jsonObject.getInt("edad");
    // Objeto anidado
    JSONObject direccion = jsonObject.getJSONObject("direccion");
    String ciudad = direccion.getString("ciudad");
    String codigoPostal = direccion.getString("codigoPostal");
    // Array con elementos mixtos
    JSONArray hobbies = jsonObject.getJSONArray("hobbies");
    String hobbie1 = hobbies.getString(0);
    String hobbie2 = hobbies.getString(1);
    JSONObject hobbie3 = hobbies.getJSONObject(2);
    String hobbie3Nombre = hobbie3.getString("nombre");
    String hobbie3Nivel = hobbie3.getString("nivel");
    // Mostrar en log
    Log.d("JSON", "Nombre: " + nombre + ", Edad: " + edad);
    Log.d("JSON", "Ciudad: " + ciudad + ", Código Postal: " + codigoPostal);
    Log.d("JSON", "Hobbies:");
    Log.d("JSON", "- " + hobbie1);
    Log.d("JSON", "- " + hobbie2);
    Log.d("JSON", "- " + hobbie3Nombre + " (" + hobbie3Nivel + ")");
} catch (JSONException e) {
    e.printStackTrace();
}
```

Ejercicio 1

- Crea un proyecto nuevo, con las siguientes características:
  - Nombre: “9 Assets”
  - Paquete: dam.assets
  - Directorio: “9-Assets”
- Implementa una aplicación que lea el contenido de un fichero que se encuentre en el directorio *res/raw* y lo muestre por pantalla (debe permitir scroll, si el texto es grande, y respetar los saltos de línea)
- También debe leer el contenido de un fichero JSON, que se encuentre en el directorio *assets*, y muestre por pantalla el valor de sus propiedades.
- El fichero JSON se llama *Configuración.json* y su contenido es :

```
{
   "retraso-klm" : {
        "descarga": 5,
        "análisis": 30
   }
}
```



Analizar un fichero XML

- Hay diferentes tipos de analizadores XML, por ejemplo SAX o DOM.
- En un analizador DOM, se carga el documento en memoria y se manipula desde allí.
- En SAX la iteración del XML la realiza el analizador generando eventos *push*, que deben ser capturados, según va encontrando elementos. Es adecuado para ficheros grandes, pues no los almacena en memoria durante el procesamiento
- Android dispone de clases tanto para SAX como para DOM y también una implementación similar a StAX (variante de SAX donde el programa solicita de forma explícita la lectura del siguiente elemento para responder a su evento *pull*)

Clases e interfaces usados en SAX

Se usan al menos dos clases y una interfaz para analizar fichero XML con un analizador SAX:

- SAXParserFactory. Permite obtener un objeto `SAXParserFactory` mediante el método `newInstance()`
- SAXParser. Objeto cuya instancia se obtiene mediante el método `newSAXParser()` de un objeto `SAXParserFactory`
- ContentHandler. Interfaz que define los métodos que se deben implementar en la clase que se va a usar para tratar los eventos

Analizar un XML con SAX (1)

```
try {
  final SAXParserFactory fabrica = SAXParserFactory.newInstance();
  fabrica.setNamespaceAware(true); // Soporte de espacios de nombres
  SAXParser analizadorSAX = fabrica.newSAXParser();
  // AnalizadorXML es la clase que implementa ContentHandler
  final AnalizadorXML analizadorXML = new AnalizadorXML();
  final InputStream inputStream = getAssets().open("fichero.xml");
  analizadorSAX.parse(inputStream, analizadorXML);
 // ¿Cómo recoger el resultado del análisis?
} catch (SAXException | IOException | ParserConfigurationException e) {
   Log.d("Errores SAX", "Se ha producido un error: " + e.toString());
}
```

Analizar un XML con SAX (2)

```
class AnalizadorXML extends DefaultHandler {
  @Override
  public void startDocument() throws SAXException {
        super.startDocument();
  }
  @Override
  public void startElement(String namespaceURI, String nombreLocal, String nombreCualif, Attributes atributos) throws SAXException {
        super.startElement(namespaceURI, nombreLocal, nombreCualif, atributos);
  }
  @Override
   public void characters(char ch[], int comienzo, int longitud) throws SAXException{
        super.characters(ch, comienzo, longitud);
  }
  @Override
  public void endElement(String namespaceURI, String nombreLocal, String nombreCualif) throws SAXException {
        super.endElement(namespaceURI, nombreLocal, nombreCualif);
  }
  @Override
  public void endDocument() throws SAXException {
        super.endDocument();
  }
}
```

Estructura de un fichero KML

KML (**K**eyhole **M**arkup **L**anguage): formato de documento XML usado para representar datos geográficos en tres dimensiones. Ejemplos:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
 <Placemark>
     <name>Nombre del primer elemento</name>
     <description>Descripción del elemento primero.</description>
     <Point>
        <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
     </Point>
 </Placemark>
  <Placemark>
     <name>Nombre del segundo elemento</name>
     <description>Descripción del elemento segundo.</description>
     <Point>
        <coordinates>-142.0822035425683,38.42228990140251,0</coordinates>
     </Point>
 </Placemark>
</kml>
```

```xml
<kml xmlns="http://earth.google.com/kml/2.2">
    <Placemark>
      <description>http://informo.munimadrid.es/cameras/Camara06303.jpg</des
      <styleUrl>#m_ylw-pushpin</styleUrl>
      <ExtendedData>
        <Data name="Numero">
          <Value>06303</Value>
        </Data>
        <Data name="Nombre">
          <Value>PLAZA DE CASTILLA (NORTE)</Value>
        </Data>
      </ExtendedData>
      <Point>
        <altitudeMode>relativeToGround</altitudeMode>
        <coordinates>-3.68894207537291,40.466063829633,10 </coordinates>
      </Point>
    </Placemark>
    <Placemark>
      <description>http://informo.munimadrid.es/cameras/Camara06304.jpg</des
      <styleUrl>#m_ylw-pushpin</styleUrl>
      <ExtendedData>
        <Data name="Numero">
          <Value>06304</Value>
        </Data>
        <Data name="Nombre">
          <Value>PLAZA DE CASTILLA (SUR)</Value>
        </Data>
      </ExtendedData>
      <Point>
        <altitudeMode>relativeToGround</altitudeMode>
        <coordinates>-3.68963237924212,40.4657761535436,10 </coordinates>
      </Point>
    </Placemark>
    <Placemark>
```

https://developers.google.com/kml/documentation/kml_tut?hl=es

Ejemplo de analizador SAX para un KML (1)

Objetivo: obtener el nombre de cada cámara a partir de un documento KML con datos de cámaras

```
protected void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   String resultado = procesarXML();
   TextView tvCont=findViewById(R.id.TVasset);
   tvCont.setText(resultado);
}
```


```
private String procesarXML() {
    String resultado;
    try {
     final SAXParserFactory fab = SAXParserFactory.newInstance();
     fab.setNamespaceAware(true);
      SAXParser analizadorSAX = fabrica.newSAXParser();
      final AnalizadorXML analizadorXML = new AnalizadorXML();
      final InputStream is = getAssets().open("CCTV.kml");
      analizadorSAX.parse (is, analizadorXML);
      resultado = analizadorXML.getResultado();
    } catch (SAXException | IOException
                | ParserConfigurationException e) {
            resultado="Se ha producido este error:\n" + e.toString();
   }
    return resultado;
  }
```

Ejemplo de analizador SAX para un KML (II)

Ejemplo de analizador SAX para un KML (3)

```
@Override
public void endElement(String namespaceURI, String nombreLocal, String nombreCualif) throws SAXException {
  super.endElement(namespaceURI, nombreLocal, nombreCualif);
  switch (nombreLocal) {  // Procesar los elementos que interesan (normalmente los que contienen texto)
    case "Placemark":
      // Cuando ha llegado aquí, ya se tienen los datos
      resultado.append("Nombre: " + nombre + "\n\n");
      break;
    case "Value": // Elemento que contiene texto
      if (esNombre) {
        nombre = contenido.toString().trim();
        esNombre = false;
      }
      break;
    case "description":
      break;
    ......
  }
}
@Override
public void endDocument() throws SAXException {
  super.endDocument();
}
}
```

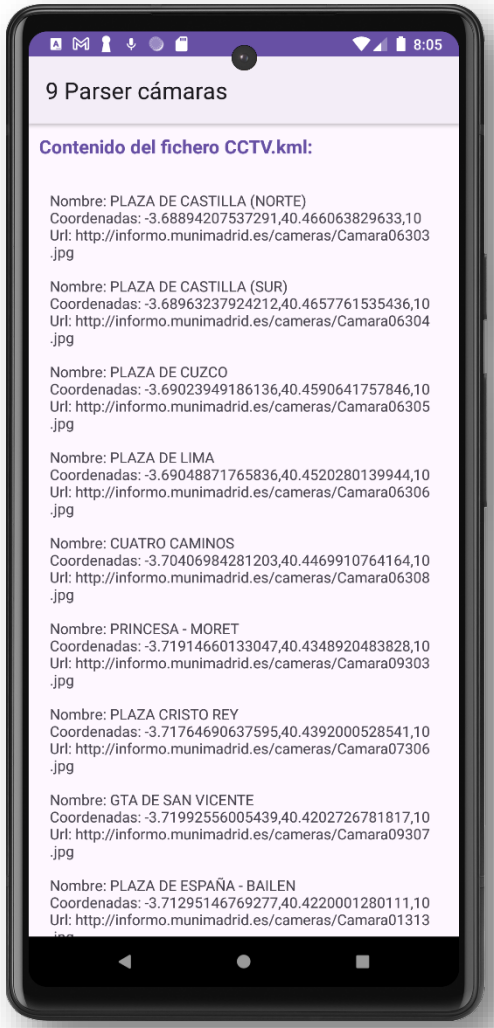
```
<kml xmlns="http://earth.google.com/kml/2.2">
    <Placemark>
      <description>http://informo.munimadrid.es/cameras/Camara06303.jpg</des
      <styleUrl>#m_ylw-pushpin</styleUrl>
      <ExtendedData>
        <Data name="Numero">
          <Value>06303</Value>
        </Data>
        <Data name="Nombre">
          <Value>PLAZA DE CASTILLA (NORTE)</Value>
        </Data>
      </ExtendedData>
      <Point>
        <altitudeMode>relativeToGround</altitudeMode>
        <coordinates>-3.68894207537291,40.466063829633,10 </coordinates>
      </Point>
    </Placemark>
    <Placemark>
      <description>http://informo.munimadrid.es/cameras/Camara06304.jpg</des
      <styleUrl>#m_ylw-pushpin</styleUrl>
      <ExtendedData>
        <Data name="Numero">
          <Value>06304</Value>
        </Data>
        <Data name="Nombre">
          <Value>PLAZA DE CASTILLA (SUR)</Value>
        </Data>
      </ExtendedData>
      <Point>
        <altitudeMode>relativeToGround</altitudeMode>
        <coordinates>-3.68963237924212,40.4657761535436,10 </coordinates>
      </Point>
    </Placemark>
    <Placemark>
```

Ejercicio 2

- Crea un proyecto nuevo, con las siguientes características:
  - Nombre: “9 Parser cámaras”
  - Paquete: dam.parsercamaras
  - Directorio: “9-ParserCamaras”

- Implementa una aplicación que lea un fichero KML (guardado en el directorio *assets*) con información de las cámaras de tráfico del ayuntamiento de Madrid y muestre, en un TextView con scroll, los datos obtenidos de las cámaras (nombre coordenadas y url)
- Puedes usar el fichero KML de moodle para este ejercicio, que es una versión modificada (únicamente con la información de las cámaras) y reducida de:

```
https://informo.madrid.es/informo/tmadrid/CCTV.kml
```



Sistema de almacenamiento persistente

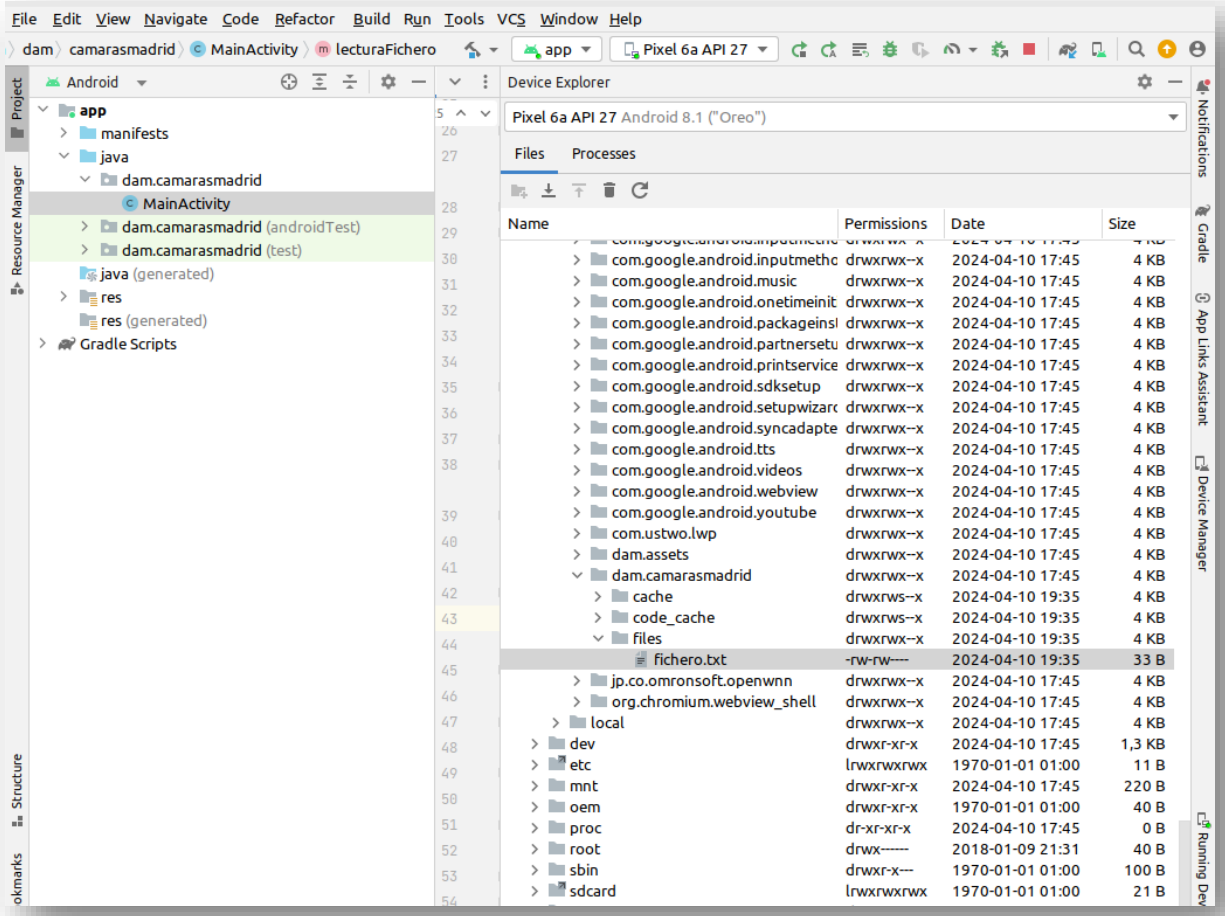
- Una aplicación puede usar ficheros almacenados en el directorio *Assets* o en el directorio *res/raw*. Pero estos ficheros deben haber sido guardados previamente antes de instalar la aplicación en el terminal, pues se copian al fichero de instalación *apk*. No son por tanto directorios de escritura durante la ejecución de la aplicación.
- Para poder almacenar ficheros en tiempo de ejecución, Android dispone de varias alternativas: almacenamiento en la memoria interna, en una tarjeta SD si existe, en una base de datos, en la nube …

Almacenamiento en memoria interna

- El sitio más sencillo para almacenar ficheros, que no requiere de petición de permisos explícitos al usuario, es la memoria interna:
  - Los ficheros se guardarán en el directorio `/data/data/paquete/files/`
  - Es posible crear subdirectorios a partir de ahí.
  - Se puede conocer el path donde se guardan los ficheros de una aplicación con el método `getFilesDir()`.
  - En Android Studio puede verse el directorio con el *Device File Explorer*.
- En esta guía se exponen los diferentes sistemas de almacenamiento.

Almacenamiento en memoria interna

/data/data/dam.camarasmadrid/files/fichero.txt



Escribir fichero en almacenamiento interno

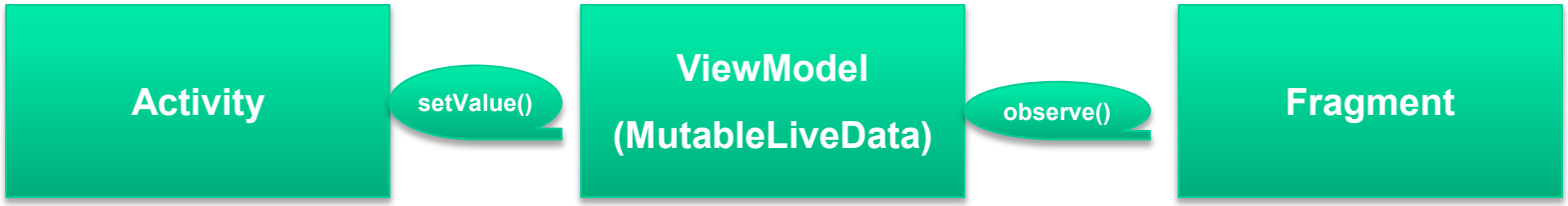
```
private void escrituraFichero(String nombreFichero) {
   try {
      final FileOutputStream fileOutStream = openFileOutput(nombreFichero, Context.MODE_PRIVATE);
      final BufferedWriter buffWriter = new BufferedWriter(new OutputStreamWriter(fileOutStream));
      buffWriter.write("Contenido a guardar en el fichero");
      buffWriter.close();
      Log.d ("Mensaje","El fichero se ha guardado en "+getFilesDir()+ File.separator+nombreFichero);
   } catch (IOException e) {
      Log.e ("Mensaje","Error al guardar el fichero "+e.getMessage());
   }
}
```

Leer fichero de almacenamiento interno

```
private String lecturaFichero(String nombreFichero){
   StringBuilder respuesta=new StringBuilder();
   String linea;
   try {
      final InputStream inputStream = openFileInput(nombreFichero);
      final BufferedReader buffReader = new BufferedReader(new InputStreamReader(inputStream));
      while ((linea = buffReader.readLine()) != null) {
         respuesta.append(linea + "\n");
      }
      inputStream.close();
   } catch (FileNotFoundException e) {
      respuesta.append("El fichero "+nombreFichero+" no existe");
      Log.e ("Mensaje","El fichero no existe");
   } catch (IOException e) {
      respuesta.append("Error al recuperar el fichero "+nombreFichero);
      Log.e ("Mensaje","Error al recuperar el fichero "+e.getMessage());

   }
   return respuesta.toString();
}
```

LiveData

- Contenedor de datos *observable* que respeta el ciclo de vida.
- Notifica automáticamente a los observadores
- Mecanismo ideal para que una actividad notifique a un fragmento cuando tiene datos listos (o entre fragmentos)
- Suele contenerse en un atributo de un objeto ViewModel compartido entre actividad y fragmento



Ejemplo: ViewModel

```
public class DatosCompartidosViewModel extends ViewModel {
    private final MutableLiveData<List<String>> dato = new MutableLiveData<List<String>>();

    public List<String> getDato() {
        return dato.getValue();
    }

    public void setDato(List<String> listado) {
        dato.postValue(listado);
    }
}
```

Ejemplo: Actividad

```
public class MainActivity extends AppCompatActivity {

   private DatosCompartidosViewModel datosCompartidos;

   @Override
   protected void onCreate (Bundle savedInstanceState) {

         :::::::
         datosCompartidos = new ViewModelProvider(this).get(DatosCompartidosViewModel.class);
         :::::::

         dato=generarDatos(); // Generar datos, esto llevará un tiempo y lo debería hacer un hilo
         datosCompartidos.setDatos(dato); // Almacenar dato

   }
}
```

Ejemplo: fragmento

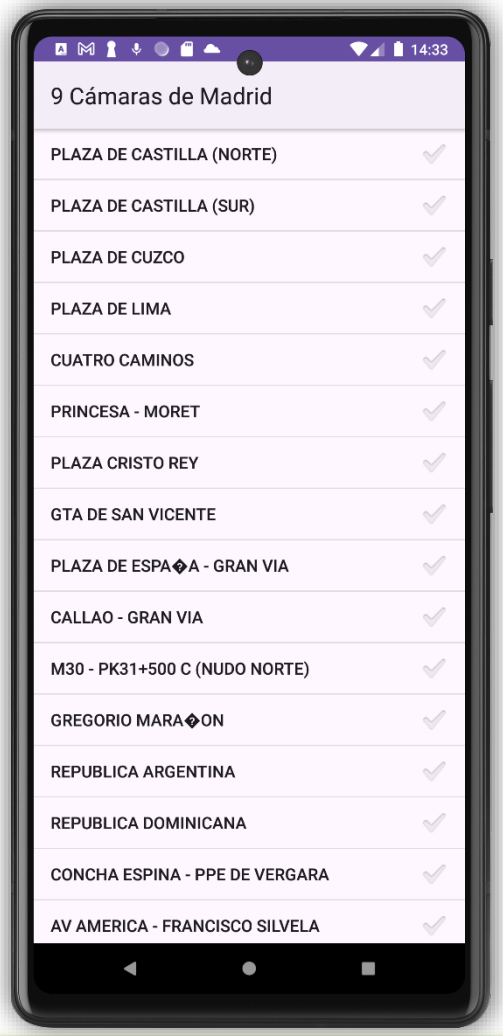
```
public class Fragmento extends Fragment {

   private DatosCompartidosViewModel datosCompartidos;

   @Override
   protected void onAttach (Context context) {
      :::::::
      datosCompartidos = new ViewModelProvider(getActivity()).get(DatosCompartidosViewModel.class);
      :::::::
   }

   @Override
   protected void onViewCreated (View vista, Bundle savedInstanceState) {
      :::::::
      datosCompartidos.getDato().observe(getViewLifecycleOwner(), new Observer<List<String>>() {
          @Override
          public void onChanged(final List<String> nuevosDatos) {
              usarDatos(nuevosDatos);
          }
       });
      :::::::
   }
}
```

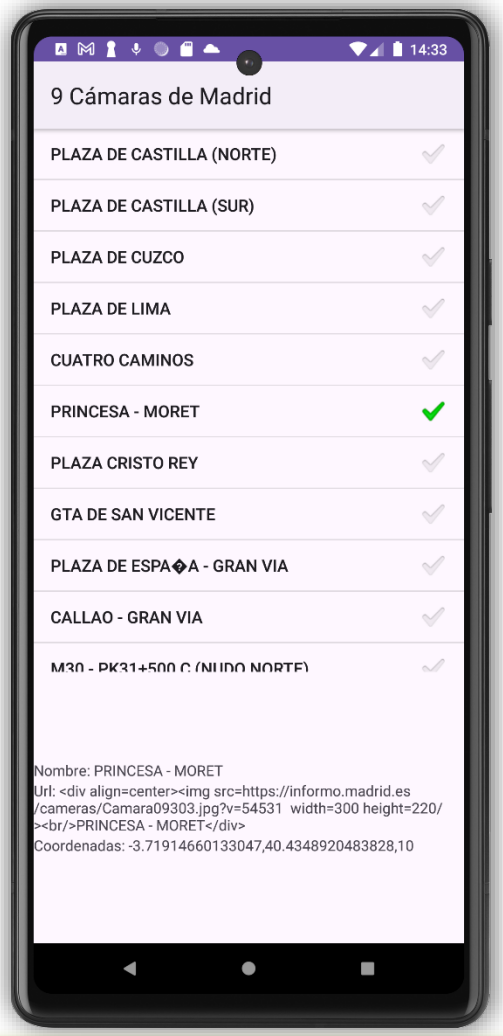
Ejercicio 3 (1)

- Crea un proyecto nuevo, con las siguientes características:
  - Nombre: “9 Cámaras de Madrid”
  - Paquete: dam.camarasmadrid
  - Directorio: “9-CamarasMadrid”
- Basándote en el proyecto del ejercicio anterior, crea una aplicación con las siguientes funcionalidades:
  - Al iniciarse, descarga de internet el fichero klm con las cámaras y mientras lo hace muestra una barra de progreso horizontal que avance con las líneas leídas
  - Tras descargar el klm realiza el análisis para obtener las cámaras. Mientras lo hace muestra un contador con las cámaras extraídas hasta el momento y una barra de progreso circular.
  - Tras disponer de los datos de todas las cámaras, las muestra en una lista ListView que ocupa toda la pantalla
  - Al seleccionar una cámara muestra, en la parte inferior (o a la derecha según la orientación), los siguientes datos de la cámara seleccionada: nombre, URL y la localización geográfica.

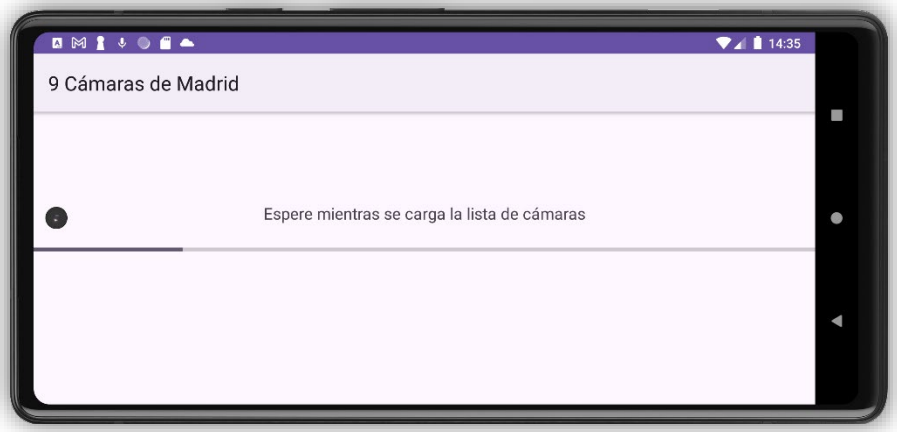
Ejercicio 3 (2)

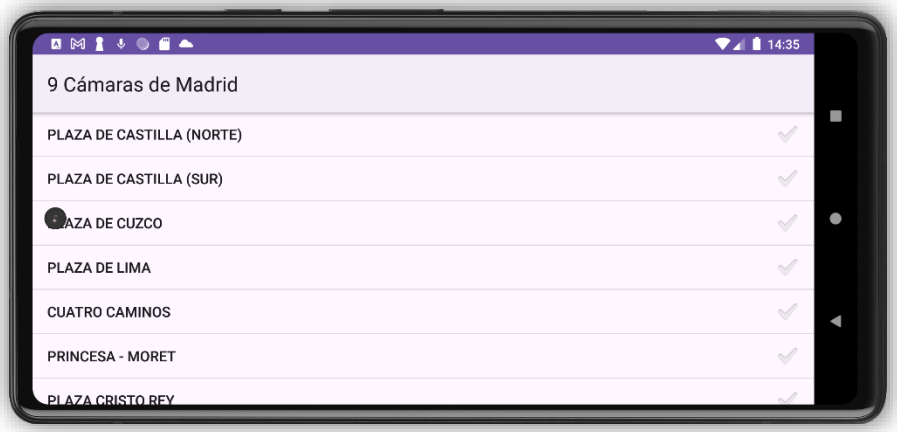
Ejercicio 3 (3)

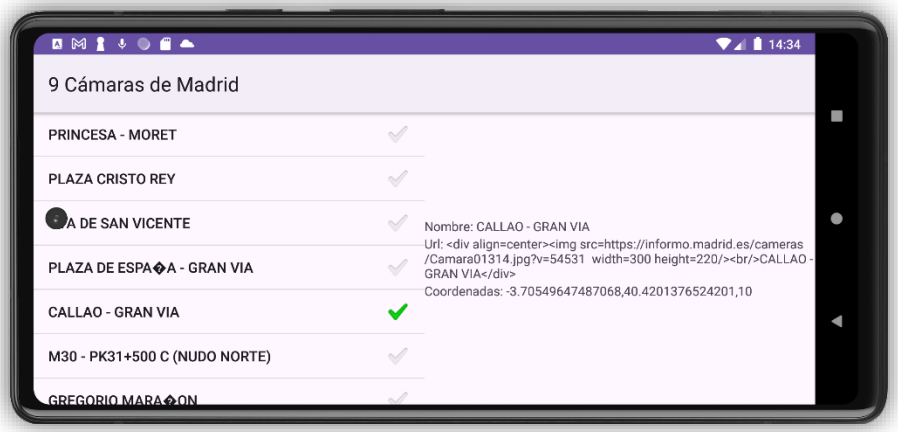




Ejercicio 3 (4)







Ejercicio 3 (5)

Restricciones:

- Crea un paquete llamado `modelo`, dentro de `dam.parsercamarasmadrid`, con:
  - La clase del listado de cámaras
  - La clase que representa una cámara (clase POJO)
  - La clase del analizador XML que implementa `DefaultHandler`
  - La clase con el hilo de trabajo que ejecuta la clase del analizador
  - La clase con el hilo de trabajo que descarga el fichero KLM
- Crea dos fragmentos:
  - Fragmento listado estático: realizará el análisis y mostrará la lista de cámaras
  - Fragmento detalle dinámico: mostrará los datos de la cámara seleccionada
- Implementa la descarga del fichero klm en un hilo de trabajo. Este hilo lo debe lanzar la actividad principal y se encargará de hacer la descarga y guardar los datos descargados en un fichero en el almacenamiento interno.
- Implementa el analizador XML en un hilo de trabajo. Este hilo lo debe lanzar el fragmento listado y se encargará de obtener las cámaras del fichero descargado. Cuando termine, ocultará esos elementos y se lo comunicará al fragmento listado para que actualice el ListView y lo haga visible
- El fragmento listado debe ocupar toda la pantalla hasta que se haga una selección. En ese momento es cuando se mostrará el fragmento detalle
- Para poder ver el progreso mete retardos en la descarga y en el análisis. Los valores de los retardos se deben leer de un documento JSON como el que se usó en el ejercicio 1 (el valor 0 indica que no hay retardos)
- Se debe guardar el estado en el cambio de orientación:
  - Si ya existe un fichero descargado, entonces no debe hacer la descarga de nuevo
  - Si ya se ha seleccionado una cámara, al cambiar de orientación debe mostrarse sus datos y mantener la selección
- Pon un icono a la aplicación

Ejercicio 3 (6)

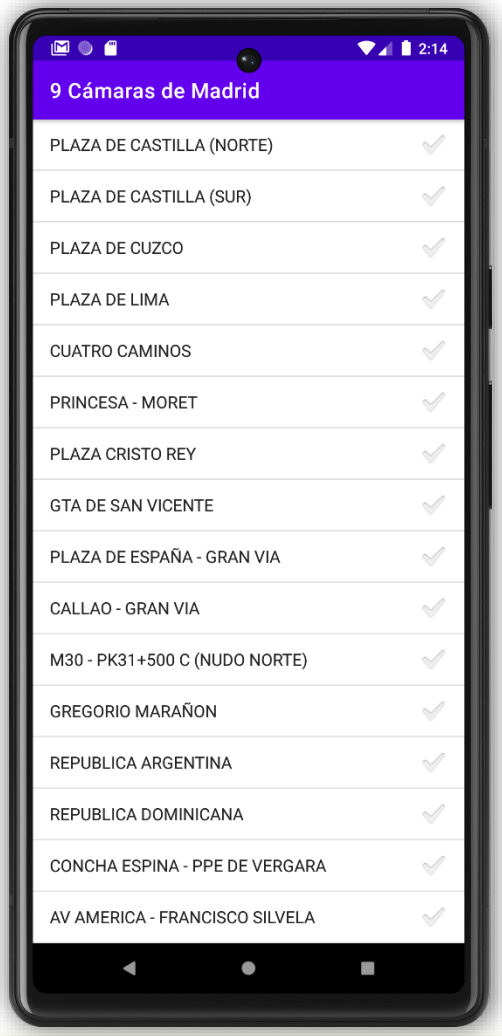
Pistas:

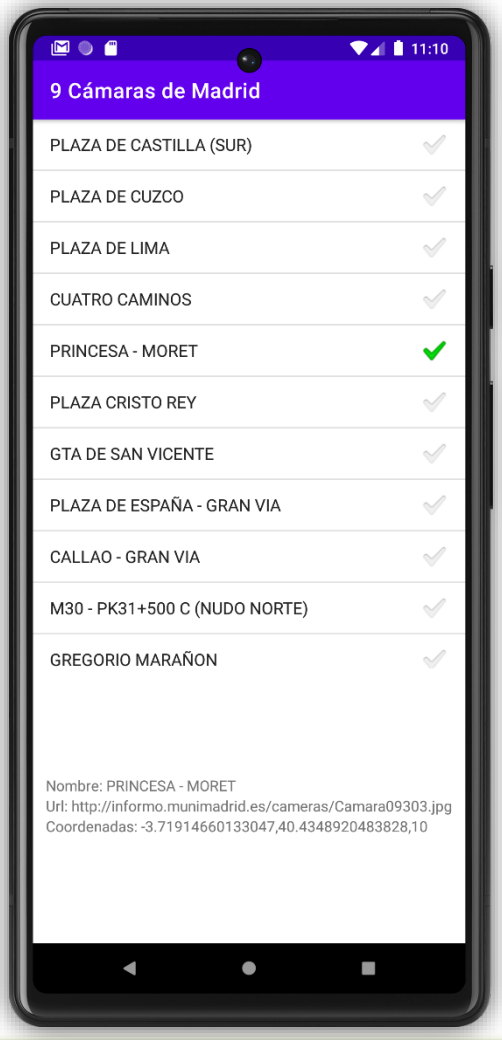
- En el diseño de la actividad principal incluye los contenedores para el fragmento listado y detalle. Si tienen pesos distintos (`layout_weight`) en la orientación vertical puedes conseguir el efecto de que, cuando no hay selección y el fragmento detalle esté oculto, el fragmento listado ocupe toda la pantalla.
- Comienza usando el fichero KML de moodle que tiene el listado completo de cámaras sin elementos adicionales
- Puedes partir del código de la aplicación “Listado de países con fragmentos” realizada en la sesión 7, teniendo en cuenta que no se necesita un adaptador personalizado
- Puedes incorporar una versión reducida de la clase de la tarea asíncrona, de la sesión 6, para descargar el fichero KLM
- Usa un atributo booleano para que, cuando esté a true, el hilo que realiza el análisis meta un retardo cada vez que actualiza el contador de cámaras procesadas. De esta manera puedes controlar si quieres ver el progreso lentamente o no, pues el análisis del xml puede ser demasiado rápido. Puedes hacer lo mismo para el hilo de descarga del fichero klm
- Como el fragmento listado es estático (se define en el fichero de diseño de la actividad y se instancia al crear la actividad), debe esperar a que la actividad descargue el fichero klm y le avise cuando termine
- Como el fragmento detalle es dinámico, se debe crear (si no existe), cuando se ha hecho una selección
- En una clase fragmento no tienes disponible el método `getAssets()`. Usa `getContext().getAssets()`
- Usa una clase `ViewModel` con un atributo booleano para saber si se ha realizado la descarga del fichero klm, para que, en un cambio de orientación, no tener que descargarlo de nuevo y así usar el fichero descargado (guardado en el sistema de almacenamiento interno). Usa un objeto LiveData para este atributo para que el fragmento listado observe cuando la actividad ha terminado de descargar el fichero klm y así poder lanzar el hilo de análisis
- En esta clase `ViewModel` usa un atributo para guardar el objeto con el listado de cámaras, y de esta manera no se tenga que realizar el análisis del fichero KML con cada cambio de orientación. Usa un objeto LiveData para este atributo para que el fragmento listado observe cuando ha terminado el análisis y así poder mostrarlo
- Si guardas el objeto con listado de cámaras en el objeto `ViewModel` en cuanto se obtiene, no hace alta que también lo hagas en el método `onSaveInstanceState()` que se ejecuta antes de que se destruya el fragmento con un cambio de orientación. Pero para recuperarlo no puedes usar el método `onRestoreInstanceState()`, porque no existe en la clase `Fragment`. Recupéralo en `onCreate()`(o en `onViewCreated()` si hay que modificar elementos de diseño) consultando si existe el `Bundle` del argumento `savedInstanceState`

Ejercicio 3 (7)

Contenedor del fragmento listado:

layout_height="0dp"
layout_weight="2"





Contenedor del fragmento detalle:

layout_height="0dp"
layout_weight="1"

Orden de ejecución de los métodos

Ten en cuenta que, cuando se cambia de orientación y se vuelven a crear los fragmentos, el orden de ejecución de los métodos de cada clase es:

- `MainActivity.super.onCreate()`
- `ListadoCamarasFragmento.onCreate()`
- `DetalleCamaraFragmento.onCreate()`
- `MainActivity.onCreate()` (resto del código)
- `ListadoCamarasFragmento.onCreateView()`
- `ListadoCamarasFragmento.onViewCreated()`
- `DetalleCamaraFragmento.onCreateView()`
- `DetalleCamaraFragmento.onViewCreated()`