Desarrollo de apps con Android

8. Conexión a Internet

Realizar una conexión a Internet

1. Añadir permisos en Android Manifest.
2. Comprobar que hay conexión de red.
3. Crear nuevo hilo de ejecución (*Worker Thread*).
4. Implementar la tarea en modo background:
    a. Crear URI.
    b. Realizar la conexión HTTP.
    c. Conectarse y obtener los datos.
5. Procesar los resultados del proceso.

1. Permisos

Permisos en AndroidManifest

- **Internet:**

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

- **Comprobar el estado de la red:**

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

2. Gestión de la conexión de red

Obtener información de la red

- ConnectivityManager
  - Responde a consultas sobre el estado de la conectividad de la red.
  - Notifica a las aplicaciones cuando cambia la conectividad de la red.
  - getActiveNetwork(): devuelve un objeto de tipo `Network` que se corresponde con la red de datos que se encuentre activa. Sin embargo, si se llegara a desconectar, entonces el objeto ya no se podrá volver a utilizar.
  - getNetworkCapabilities(Network network): devuelve null o las NetworkCapabilities de la red.
- ConnectivityManager.NetworkCallback
  - Utilizado para notificaciones sobre cambios en la red.
  - Lo deberán heredar las clases que deseen recibir notificaciones.

Comprobar si la red está disponible (1/3)

```
ConnectivityManager connMgr = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);

Network network = cnnMgr.getActiveNetwork();
NetworkCapabilities networkCap =
connMgr.getNetworkCapabilities(network);

if (networkCap!= null)) {
    // Hay conexión de red.
} else {
    // No hay conexión de red.
}
```

Comprobar si la red está disponible (2/3)

```
private ConnectivityManager.NetworkCallback networkCallback =
    new ConnectivityManager.NetworkCallback()
{
    @Override
    public void onAvailable(@NonNull Network network) {
        super.onAvailable(network);
    }
    @Override
    public void onLost(@NonNull Network network) {
        super.onLost(network);
    }
```

Sigue

Comprobar si la red está disponible (3/3)

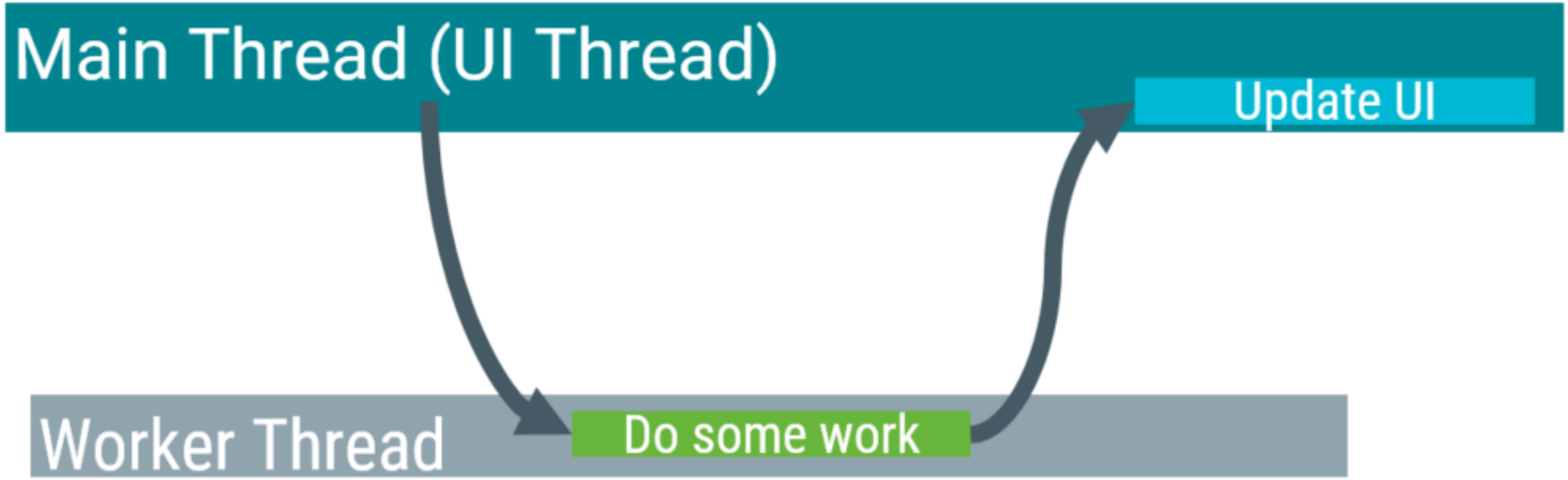
```
    @Override
    public void onCapabilitiesChanged(@NonNull Network network,
              @NonNull NetworkCapabilities networkCapabilities)
    {
      super.onCapabilitiesChanged(network, networkCapabilities);
      final boolean unmetered = networkCapabilities.hasCapability
                  (NetworkCapabilities.NET_CAPABILITY_NOT_METERED);
    }
};
```

3. Crear Worker Thread

Reglas para hilos de ejecución en Android

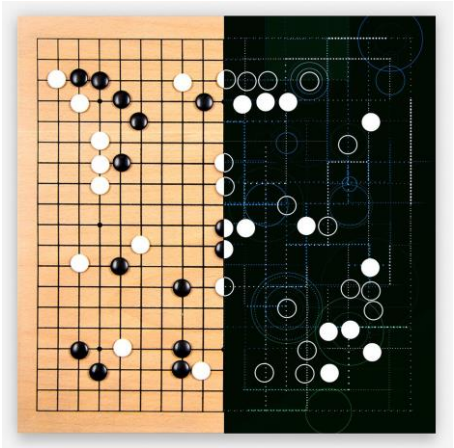
- No bloquear el **hilo de ejecución principal** (*UI Thread*):
  - Para cada pantalla se debe terminar una tarea en menos de 16 ms.
  - Toda tarea que no requiera trabajar sobre la interfaz de usuario ejecutarla en un nuevo hilo de ejecución.
- No acceder al Android UI toolkit desde fuera del *UI Thread*:
  - Las tareas de de interfaz de usuario solamente se han de realizar en el *UI Thread*.

Creación de hilos de ejecución (Worker Thread)



¿Cuándo considerar tareas background?

- Operaciones de red.
- Cálculos complejos de larga duración.
- Subida y descarga de ficheros.
- Procesamiento de imágenes.
- Carga de datos.



Loader

Loader

- Proporciona carga de datos de manera asíncrona.
- Se reconecta a la `Activity` cuando cambia la configuración del dispositivo.
- Puede monitorizar cambios en la fuente de datos y entregar datos nuevos.
- Callbacks implementados en la `Activity`.
- ViewModel es otra alternativa a Loader.

LoadManager

- Gestiona las funciones loader a través de callbacks.
- Puede gestionar varios loaders: para datos en una base de datos, para datos de tareas asíncronas, para datos de Internet…

Obtener un loader con initLoader()

- Crea e inicia un loader, o reutiliza uno ya existente incluyendo sus datos.

```
loaderCallbacks = new LoaderCallbacks(this);
LoaderManager loaderManager = LoaderManager.getInstance(this);
if(loaderManager.getLoader(LOADER_ID) != null){
     loaderManager.initLoader(LOADER_ID,null, loaderCallbacks);
}
```

- Utilizar `restartLoader()` para inicializar los datos de un loader ya existente.

```
Bundle queryBundle = new Bundle();
queryBundle.putString("queryString", userQuery);
LoaderManager.getInstance(this).restartLoader(LOADER_ID, queryBundle,
loaderCallbacks);
```

Implementación AsyncTaskLoader

Pasos para implementar AsyncTaskLoader

1. Definir subclase AsyncTaskLoader
2. Implementar constructor
3. loadInBackground()
4. onStartLoading()

Subclase de AsyncTaskLoader

```
public static class StringListLoader
        extends AsyncTaskLoader<List<String>> {

    public StringListLoader(Context context, String queryString) {
        super(context);
        mQueryString = queryString;
    }
}
```

loadInBackground()

```
@Override
public List<String> loadInBackground() {
    List<String> data = new ArrayList<String>;
    //TODO: Cargar los datos desde la red o una base de datos.
    return data;
}
```

onStartLoading()

Cuando se invoca a `restartLoader()` o `initLoader()`, `LoaderManager` invoca el callback `onStartLoading()`:

- Comprueba si hay datos en caché.
- Empieza a observar la fuente de datos (si fuera necesario).
- Invoca a `forceLoad()` para cargar los datos si ha habido cambios o no hay datos en la caché.

```
protected void onStartLoading() {  forceLoad();  }
```

Implementar LoaderCallbacks en la Activity

- `onCreateLoader()`: Crear un nuevo Loader para un ID dado.
- `onLoadFinished()`: Se invoca cuando un loader que se ha creado previamente ha finalizado su carga.
- `onLoaderReset()`: Se invoca cuando un loader creado previamente se está reiniciando porque los datos no están disponibles.

onCreateLoader()

```
@Override
public Loader<List<String>> onCreateLoader(int id, Bundle args) {
    return new StringListLoader(this,args.getString("queryString"));
}
```

onLoadFinished()

El resultado de `loadInBackground()` se pasa a `onLoadFinished()` donde ya se pueden visualizar en la IU:

```
@Override
public void onLoadFinished(Loader<List<String>> loader,
List<String> data) {
    mAdapter.setData(data);
}
```

onLoaderReset()

- Solamente se le invoca cuando se destruye el loader.
- Normalmente se deja en blanco.

```
@Override
public void onLoaderReset(final Loader<List<String>> loader) { }
```

4. Implementar tarea background

4.a Crear URI

Uniform Resource Identifier (URI)

String que nombra o localiza un recurso determinado:

- file://
- http:// and https://
- content://

Ejemplo de URL para Google Books API

https://www.googleapis.com/books/v1/volumes?q=pride+prejudice&maxResults=5&printType=books

Constantes para parámetros:

```
final String BASE_URL =
    "https://www.googleapis.com/books/v1/volumes?";
final String QUERY_PARAM = "q";
final String MAX_RESULTS = "maxResults";
final String PRINT_TYPE = "printType";
```

Construir la URI para la petición

```
Uri builtURI = Uri.parse(BASE_URL).buildUpon()
        .appendQueryParameter(QUERY_PARAM, "pride+prejudice")
        .appendQueryParameter(MAX_RESULTS, "10")
        .appendQueryParameter(PRINT_TYPE, "books")
        .build();
URL requestURL = new URL(builtURI.toString());
```

4.b Conexión cliente HTTP

Realizar una conexión desde cero

- Utilizar HttpURLConnection.
- Se debe realizar en un hilo (thread) independiente.
- Requiere InputStreams y bloques try/catch.

Crear HttpURLConnection

```
HttpURLConnection conn =
  (HttpURLConnection) this.requestURL.openConnection();
```

Configurar conexión

```
conn.setReadTimeout(10000 /* milliseconds */);
conn.setConnectTimeout(15000 /* milliseconds */);
conn.setRequestMethod("GET");
conn.setDoInput(true);
```

Realizar conexión y obtener respuesta

```
conn.connect();
int response = conn.getResponseCode();

InputStream is = conn.getInputStream();
String contentAsString = convertIsToString(is, len);
return contentAsString;
```

Cerrar conexión y stream

```
} finally {
        conn.disconnect();
         if (is != null) {
             is.close();
         }
}
```

4.c Transformar la respuesta a String

Transformar InputStream en un string

```
public String convertIsToString(InputStream stream, int len)
    throws IOException, UnsupportedEncodingException {

    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    char[] buffer = new char[len];
    reader.read(buffer);
    return new String(buffer);
}
```

BufferedReader es más eficiente

```
StringBuilder builder = new StringBuilder();
BufferedReader reader =
    new BufferedReader(new InputStreamReader(inputStream));
String line;
while ((line = reader.readLine()) != null) {
   builder.append(line + "\n");
}
if (builder.length() == 0) {
   return null;
}
resultString = builder.toString();
```

Librerías externas para conexión con cliente HTTP

Realizar una conexión utilizando librerías externas

- Utilizar librerías de terceros como OkHttp , Volley, Retrofit o Ktor (*Kotlin coroutines*).
- Se pueden invocar desde el hilo principal.
- Supone codificar menos.
- También se pueden realizar peticiones con Cronet.

OkHttp

```
OkHttpClient client = new OkHttpClient();
Request request = new Request.Builder()
    .url("http://publicobject.com/helloworld.txt").build();
client.newCall(request).enqueue(new Callback() {
    @Override
    public void onResponse(Call call, final Response response)
        throws IOException {
        try {
            String responseData = response.body().string();
            JSONObject json = new JSONObject(responseData);
            final String owner = json.getString("name");
        } catch (JSONException e) {}
    }
});
```

Volley

```
RequestQueue queue = Volley.newRequestQueue(this);
String url ="https://www.google.com";

StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        // Do something with response
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {}
});
queue.add(stringRequest);
```

5. Procesar el resultado

Procesar el resultado

- La respuesta normalmente se da en JSON o XML.
- Procesar el resultado haciendo uso de las clases “helper”:
  - JSONObject, JSONArray
  - XMLPullParser

JSON

```
{
 "population":1,252,000,000,
 "country":"India",
 "cities":["New Delhi","Mumbai","Kolkata","Chennai"]
}
```

JSONObject

```
JSONObject jsonObject = new JSONObject(response);

String nameOfCountry = (String) jsonObject.get("country");
long population = (Long) jsonObject.get("population");
JSONArray listOfCities = (JSONArray)
jsonObject.get("cities");

Iterator<String> iterator = listOfCities.iterator();
while (iterator.hasNext()) {
    // procesar elemento
}
```

Otro ejemplo JSON (1/2)

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}
```

Otro ejemplo JSON (2/2)

Obtener el valor "onclick" del tercer elemento del array "menuitem".

```
JSONObject data = new JSONObject(responseString);
JSONArray menuItemArray =
        data.getJSONArray("menuitem");
JSONObject thirdItem =
        menuItemArray.getJSONObject(2);
String onClick = thirdItem.getString("onclick");
```