Programación multimedia y dispositivos móviles

Servicios Web

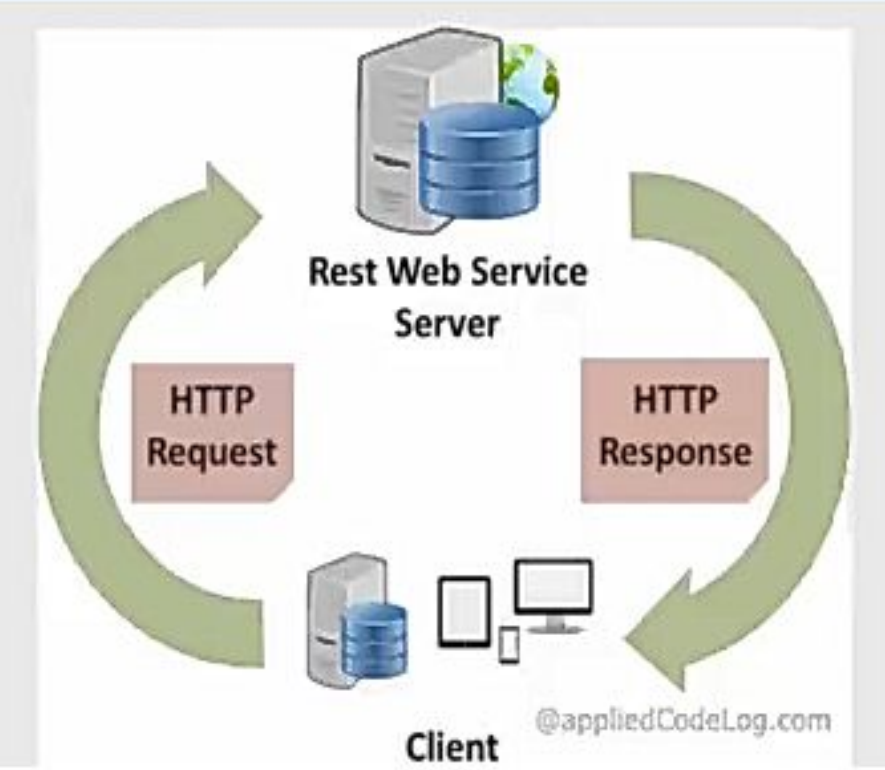
Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

Web Service

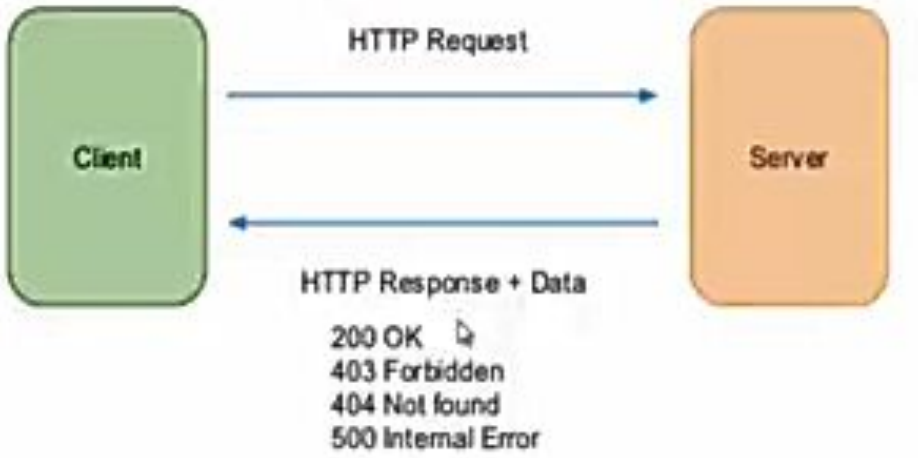
- Un web service o servicio web es una aplicación que vive en internet y que sirve para intercambiar datos entre aplicaciones.

- Debe manejar estándares de comunicación que permitan el envío y recepción de información

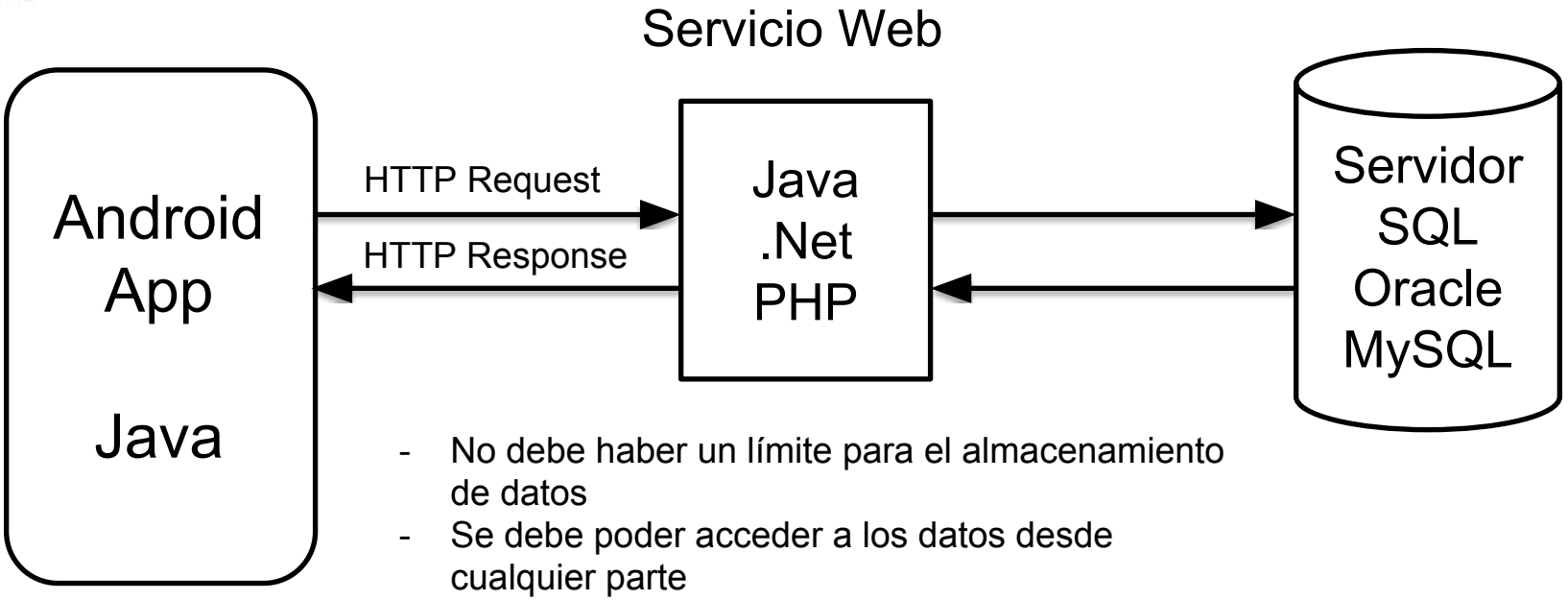


Web Service

- Cuando hablamos de consumir un Servicio Web básicamente hablamos de hacer una petición a un servidor y esperar una respuesta



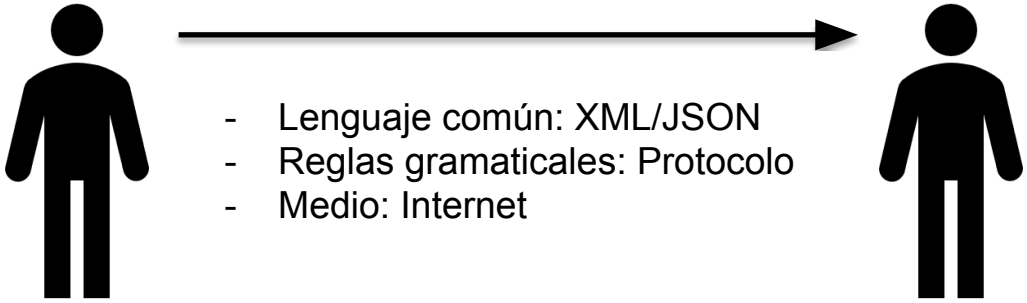
Cómo funciona?



- No debe haber un límite para el almacenamiento de datos
- Se debe poder acceder a los datos desde cualquier parte

SOAP

REST



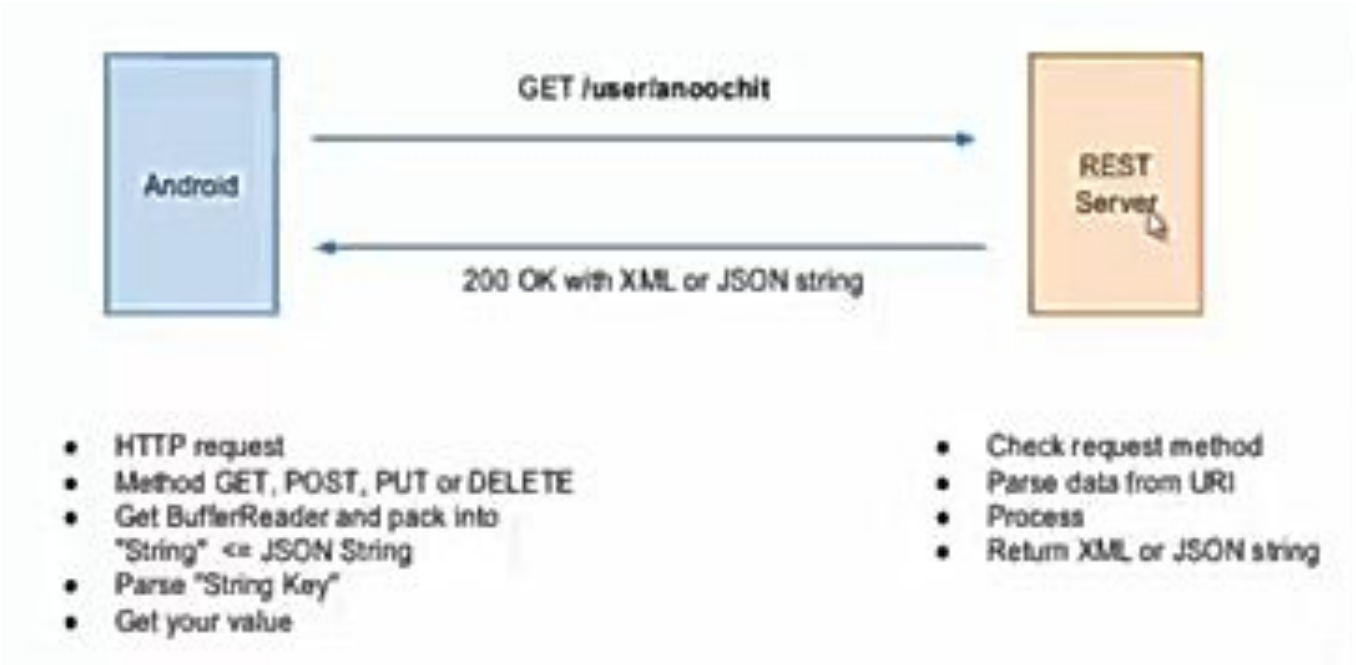
- Lenguaje común: XML/JSON
- Reglas gramaticales: Protocolo
- Medio: Internet

Emisor Java

Receptor .Net

Arquitectura REST o SOAP

- Algunos de estos estándares de comunicación que podemos manejar los encontramos en los formatos JSON y XML, en función del formato manejaremos una arquitectura REST o SOAP.



AsyncTask

- Cuando consumimos un servicio web lo recomendable es consumirlo en un hilo secundario porque si llegamos a tener una respuesta del servidor errónea la aplicación no se cae.
- Podríamos crear un hilo secundario pero los problemas aparecen cuando nos damos cuenta de que desde este hilo secundario no podemos hacer referencia directa a componentes que se ejecuten en el hilo principal, entre ellos los controles que forman nuestra interfaz de usuario, es decir, que desde el método run() no podríamos ir actualizando directamente estos componentes o por ejemplo, una barra de progreso.
- Android nos facilita esta tarea ofreciendo la clase **AsyncTask**.

AsyncTask

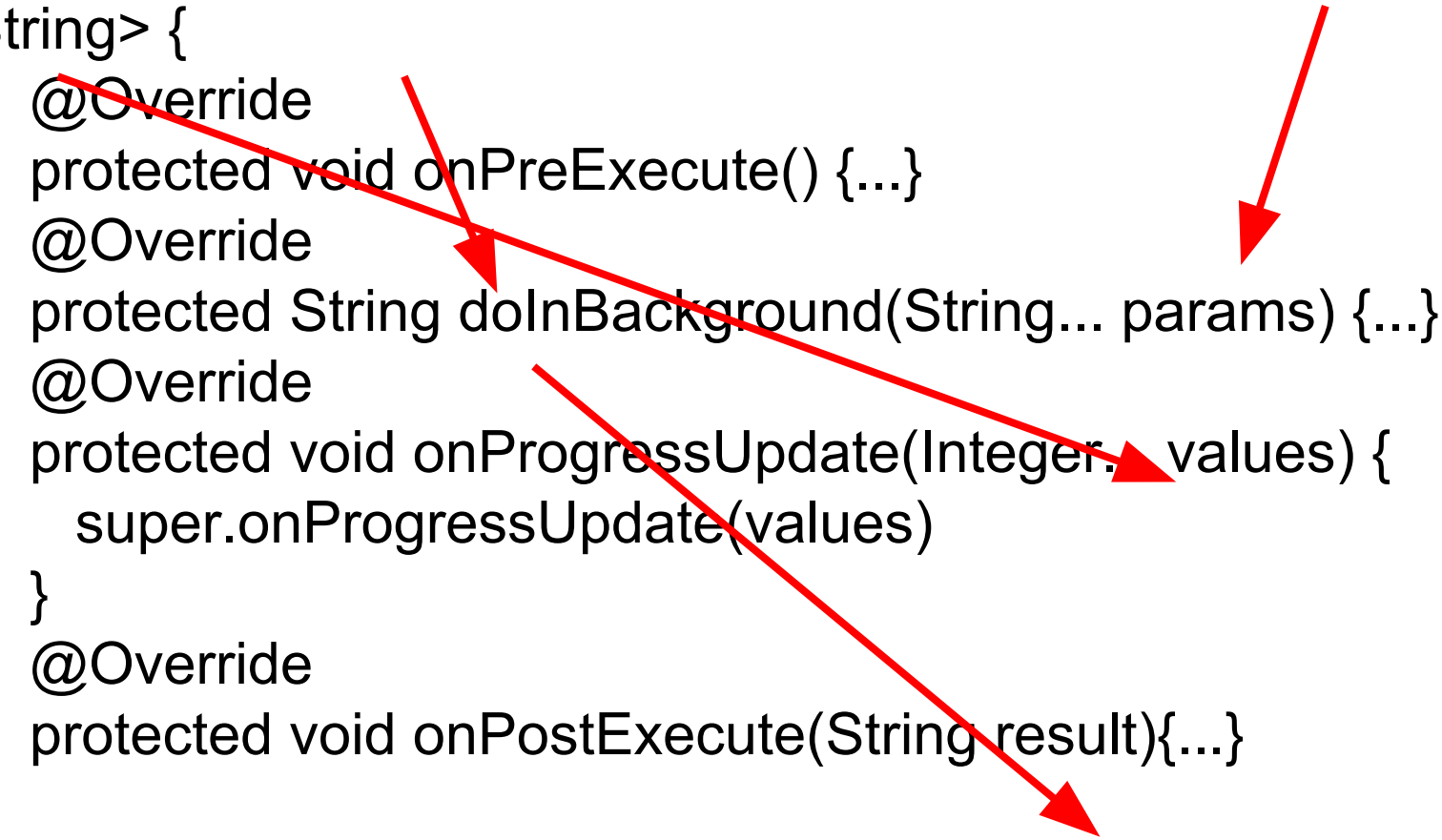
- Una AsyncTask (tarea asíncrona) representa un proceso que se ejecuta en segundo plano y cuyo resultado se publica en la interfaz de usuario.
- Queda determinada por 3 tipos genéricos que representan los parámetros, el progreso y el resultado, y 4 pasos, llamados onPreExecute, doInBackground, onProgressUpdate y onPostExecute.
- No siempre se usan todos los tipos por una tarea asincrónica. Para marcar un tipo como no utilizado, se use el tipo **Void**.

```
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

La forma básica de utilizar la clase AsyncTask consiste en crear una nueva clase que extienda de ella y sobrescribir varios de sus métodos entre los que repartiremos la funcionalidad de nuestra tarea. Estos métodos son los siguientes:

- **onPreExecute**(). Se ejecutará antes del código principal de nuestra tarea. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz, etc.
- **doInBackground**(). Contendrá el código principal de nuestra tarea.
- **onProgressUpdate**(). Se ejecutará cada vez que llamemos al método publishProgress() desde el método doInBackground().
- **onPostExecute**(). Se ejecutará cuando finalice nuestra tarea, o dicho de otra forma, tras la finalización del método doInBackground().

```
private class MiTarea extends AsyncTask<String, Integer,
String> {
    @Override
    protected void onPreExecute() {...}
    @Override
    protected String doInBackground(String... params) {...}
    @Override
    protected void onProgressUpdate(Integer... values) {
      super.onProgressUpdate(values)
    }
    @Override
    protected void onPostExecute(String result){...}
}
```



Cómo conectar con el servicio y solicitar información

```
URL url = null;
HttpURLConnection conn = null;
BufferedReader br = null;
try {
    url = new URL("url");
    conn = (HttpURLConnection) url.openConnection();
    conn.setDoInput(true);
    br = new BufferedReader(new
InputStreamReader(conn.getInputStream()));

    StringBuffer sb = new StringBuffer();
    String line = null;
    // Read Server Response
    while((line = br.readLine()) != null) {
        sb.append(line + " ");
    }
    res = sb.toString();

} catch (MalformedURLException e) {
} catch (IOException e) {}
```

Ejemplo

- Creamos un nuevo proyecto llamado App Web Service.

- Para poder conectar desde una aplicación Android a un servicio web lo primero que tendremos que hacer es especificar los permisos correspondientes en el AndroidManifest:

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

Ejemplo

- Al activity_main.xml le damos el siguiente formato:

Ejemplo

- En el MainActivity.java creamos una clase interna:

```
private class EjecutarWS extends AsyncTask<String, Void, Void> {
    private String res;
    TextView tvRes = (TextView)findViewById(R.id.tvResultado);

    @Override
    protected Void doInBackground(String... strings) {
        URL url = null;
        HttpURLConnection conn = null;
        BufferedReader br = null;
        try {
            url = new URL(strings[0]);
            conn = (HttpURLConnection) url.openConnection();
            conn.setDoInput(true);
            br = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
```

Ejemplo

```
        StringBuffer sb = new StringBuffer();
        String line = null;

        // Read Server Response
        while((line = br.readLine()) != null)
        {
            sb.append(line + " ");
        }
        res = sb.toString();
    } catch (MalformedURLException e) {
    } catch (IOException e) {
    } finally {
        if (br != null)
            try {
                br.close();
            } catch (IOException e) {}
    }
    return null;
}
```

Ejemplo

```
    protected void onPostExecute(Void v) {
        tvRes.setText(res);
    }
}

public void consumirWS(View v) {
    String url = "https://pokeapi.co/api/v2/pokemon/";
    new EjecutarWS().execute(url);
}
```

Error de conexión a la red

- Cuando nuestra aplicación va a consumir un servicio web deberíamos comprobar que el terminal está conectado a la red. Para esto necesitamos implementar el siguiente método:

```
private boolean isNetworkAvailable() {
    boolean isAvailable=false;
    //Gestor de conectividad
    ConnectivityManager manager = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
    //Objeto que recupera la información de la red
    NetworkInfo networkInfo = manager.getActiveNetworkInfo();
    //Si la información de red no es nula y estamos conectados
    //la red está disponible
    if (networkInfo!=null && networkInfo.isConnected()) isAvailable=true;

    return isAvailable;

}
```

- Además de añadir un permiso nuevo en el manifest:

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

API Rest

- API: Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción
- REST (Representational State Transfer - Transferencia de Estado Representacional)
- REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

API Rest

- Si vamos a desarrollar una app Android que se va a conectar a un servidor para consumir un web service, lo que normalmente se hace es hacer una petición GET a una API representada por una URL, que lo que hará será hacer una petición al servidor, que le devolverá una respuesta. Además, como ya hemos visto, deberá hacerse desde un hilo secundario para no correr el riesgo de bloquear la aplicación.
- Lo que hemos hecho hasta ahora es Http request + AsyncTask
- Existen herramientas que se encargan de facilitar este proceso además de agilizarlo, por ejemplo Retrofit o Volley.

Volley vs Retrofit

- **Volley** es una librería desarrollada por Google para optimizar el envío de peticiones Http desde las aplicaciones Android hacia servidores externos. Este componente actúa como una interfaz de alto nivel, liberando al programador de la administración de hilos y procesos tediosos de parsing, para permitir publicar fácilmente resultados en el hilo principal.
- **Retrofit** es un cliente HTTP de tipo seguro para Android y Java desarrollado por Square. Retrofit hace sencillo conectar a un servicio web REST traduciendo la API a interfaces Java. Retrofit fue construido encima de algunas otras librerías y herramientas poderosas. Por ejemplo, hace uso de OkHttp (del mismo desarrollador) para manejar peticiones de red. Viene con soporte para librerías para convertir de objetos JSON a Java.

Nos vamos a centrar en **Retrofit** ya que destaca tanto por su velocidad, como por su documentación y gran comunidad.

- Para usar Retrofit tendremos que importar la librería correspondiente en nuestra app de Android. Añadimos la dependencia en el build.gradle(Module:app)

```
compile 'com.squareup.retrofit2:retrofit:2.3.0'
```

- Para recuperar tipos primitivos o String

```
compile 'com.squareup.retrofit2:converter-scalars:2.3.0’
```

- Añadiremos al archivo AndroidManifest.xml los permisos de Internet

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

- El siguiente paso es crear una Interfaz (colección de métodos abstractos y propiedades constantes). En la interfaz le diremos la ubicación (URL) del servicio web que se va a consumir, crearemos los métodos que vamos a usar para ejecutar peticiones HTTP tales como GET, POST, PUT y/o DELETE.

```
public interface APIRestService {
    public static final String BASE_URL = "https://...";

    @GET("final_URL")
    Call<String> nombreMetodo(Parametros);
}
```

- Para invocar a un web service en el que la URL es variable.

Ejemplo:

```
public interface APIRestService {
    public static final String BASE_URL = "https://api.github.com/";

    @GET("users/{user}/repos")
    Call<List<Repo>> listRepos(@Path("user") String user);
}
```

@Path("user"): el texto {user} pasará a ser el valor de la variable user cuando se realice la petición.

Cuando se ejecute la petición:

```
Call<List<Repo>> call = ars.listRepos("pilar");
```

- Para invocar a un web service con parámetros.

Ejemplo:

url = http://server/folder?var1=val1&var2=val2

```
public interface APIRestService {
    public static final String BASE_URL = "http://server/";

    @GET("folder")
    Call<Objeto> nombreMetodo(@Query("var1") String var1,
        @Query("var2") String var2);
}
```

@Query("var1"): con el valor de la variable var1 se formará la cadena Query string de la URL de la petición.

Cuando se ejecute la petición:

```
Call<Objeto> call = ars.nombreMetodo("val1", "val2");
```

- Creamos la clase RetrofitClient con un método que nos facilitará una instancia de la clase Retrofit, esta clase necesita una URL para construir su instancia.

```
public class RetrofitClient {
    private static Retrofit retrofit = null;

    public static Retrofit getClient(String baseUrl) {
        if (retrofit==null) {
            retrofit = new Retrofit.Builder()
                    .baseUrl(baseUrl)
    .addConverterFactory(ScalarsConverterFactory.create())
                    .build();
        }
        return retrofit;
    }
}
```

- Por último en el onCreate:
  - primero obtendremos la instancia de la clase RetrofitClient,

```
Retrofit r = RetrofitClient.getClient(APIRestService. BASE_URL);
```

  - a continuación inicializaremos una instancia del la interfaz

```
APIRestService ars = r.create(APIRestService.class);
```

  - y por último ejecutamos la petición.

```
Call<String> call = ars.nombreMetodo(parametros);
```

Retrofit. enqueue

Solo nos quedaría una cosa: hacer la petición asíncrona. Para esto utilizaremos el método **enqueue**.

- enqueue() envía de manera asíncrona la petición y notifica a nuestra aplicación con un callback cuando el servicio no retorna una respuesta. Retrofit maneja la ejecución en el hilo de fondo para que el hilo de la UI principal no sea bloqueada o interfiera con esta.

- Para usar el método enqueue(), tenemos que implementar dos métodos callback: onResponse() y onFailure(). Solo uno de estos métodos será llamado en respuesta a una petición.

Retrofit. enqueue

- **onResponse()**: Este método es llamado para una respuesta que puede ser correctamente manejada incluso si el servidor devuelve un mensaje de error. Para obtener el código de estado para que puedas manejar situaciones basadas en estos, puedes usar el método **response.code()**. También puedes usar el método **isSuccessful()** para averiguar si el código de estado está en el rango 200-300, indicando éxito.
- **onFailure()**: invocado cuando una excepción de red ocurre comunicando al servidor o cuando una excepción inesperada ocurrió manejando la petición o procesando la respuesta.

Retrofit. enqueue

```
call.enqueue(new Callback<String>() {
    @Override
    public void onResponse(Call<String> call, Response<String> response) {
        if (!response.isSuccessful()) {
            Log.i("Resultado: ", "Error" + response.code());
        } else {
            Log.i("Resultado: ", response.body().toString());
        }
    }

    @Override
    public void onFailure(Call<String> call, Throwable t) {
        Log.e("error", t.toString());
    }
});
```

Ejemplo

- Creamos un nuevo proyecto llamado App WS Retrofit.

- Especificamos los permisos correspondientes en el AndroidManifest:

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

- Incluimos las dependencias en el build.gradle(Module:app)

```
compile 'com.squareup.retrofit2:retrofit:2.3.0'
compile 'com.squareup.retrofit2:converter-scalars:2.3.0 '
```

Ejemplo

- Al activity_main.xml le damos el mismo formato que al ejemplo anterior:

Ejemplo

- Además añadimos al activity_main.xml un progress bar que se mostrará desde que se invoca al web service hasta que recibe la respuesta:

```
<ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

Ejemplo

- Creamos un paquete llamado **retrofitUtils** y creamos el interfaz **APIRestService**

```
public interface APIRestService {
    public static final String BASE_URL = "https://pokeapi.co/api/v2/";

    @GET("pokemon/")
    Call<String> obtenerPokemon();

}
```

- Y creamos la clase **RetrofitClient**

```
public class RetrofitClient {
    private static Retrofit retrofit = null;

    public static Retrofit getClient(String baseUrl) {
        if (retrofit==null) {
            retrofit = new Retrofit.Builder()
                    .baseUrl(baseUrl)
                .addConverterFactory(ScalarsConverterFactory.create())
                    .build();
        }
        return retrofit;
    }

}
```

Ejemplo

- En el **MainActivity**.java, el método **consumirWS** :

```
public void consumirWS(View v) {

    Retrofit r = RetrofitClient.getClient(APIRestService.BASE_URL);
    APIRestService ars = r.create(APIRestService.class);
    Call<String> call = ars.obtenerPokemon();

// hacemos visible el progress bar que habremos inicializado en el
// onCreate poniéndolo como oculto
    pb.setVisibility(View.VISIBLE);
    call.enqueue(new Callback<String>() {

        String res;
        TextView tvRes = (TextView)findViewById(R.id.tvResultado);
```

Ejemplo

```
        @Override
        public void onResponse(Call<String> call, Response<String> response) {
            pb.setVisibility(View.GONE); // lo escondemos
            if (!response.isSuccessful()) {
                Log.i(TAG, "Error" + response.code());
            } else {
                res = response.body().toString();
                tvRes.setText(res);
            }
        }

        @Override
        public void onFailure(Call<String> call, Throwable t) {
            pb.setVisibility(View.GONE); // lo escondemos
            Log.e("error", t.toString());
        }
    });

}
```

Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

Madrid Valencia Canarias