

Bloque III: Servicio Web REST

- REST
- Clases Java para HTTP
- Analizando respuesta JSON (Gson)
- API REST de SWAPI:
 - Introducción
 - Petición
 - Respuesta

REST

- REST (Representational State Transfer) es un estilo de arquitectura para entornos hipermedia distribuidos
- Principios REST:
 - Dar a todos los recursos un identificador
 - Vincular los recursos
 - Usar métodos estándar
 - Recursos con múltiples representaciones
 - No mantener el estado
- HTTP se adhiere bien a estos principios
 - Dar a todos los recursos un identificador: **URI**
 - Vincular los recursos: **hiperenlaces**
 - Usar métodos estándar: **GET, POST, DELETE, PUT**
 - Recursos con múltiples representaciones: **JSON, XML, text o plano**
 - No mantener el estado: **HTTP** es un protocolo **sin estado**

REST: Recursos y Representaciones

- Los recursos no salen del servidor
- Lo que podemos obtener es una representación de un recurso: JSON, XML, texto plano, JPG, PDF, ...
- Para un mismo recurso podemos tener varias representaciones. Dos estrategias para elegir:
 - Diferentes URLs:
 - <http://www.apress.com/java>
 - <http://www.apress.com/java/csv>
 - <http://www.apress.com/java/xml>
 - Misma URL y negociar contenido:

```
GET /java HTTP/1.1
User-Agent: curl/7.30.0
Host: www.apress.com
Accept: text/csv
```

```
GET /java HTTP/1.1
User-Agent: curl/7.30.0
Host: www.apress.com
Accept: application/json
```

```
GET /java HTTP/1.1
User-Agent: curl/7.30.0
Host: www.apress.com
Accept: application/xml
```

REST: Acceso Uniforme

- Una de las grandes ventajas de REST frente a otras tecnologías de servicios Web (como SOAP) es que usa una interfaz uniforme.
- Una vez que tenemos claro como referenciar recursos y como representarlos podemos usar los métodos estándares de HTTP para operar con ellos:
 - GET: consulta un recurso y obtiene una representación
 - POST: crea un nuevo recurso
 - PUT: modifica un recurso
 - DELETE: elimina un recurso

Desde el punto de vista de la asignatura un servicio REST es igual que acceder a un sitio web

Clases Java: URL

- Permite el manejo de URLs en Java.
- Métodos más importantes:
 - Constructor:
 - **URL(String urlstr)**
 - **URL(String protocol, String host, int port, String file)**
 - Consultores:
 - **getHost()**
 - **getPath()**
 - **getPort()**
 - ...
 - Crear conexión a la URL:
 - **URLConnection.openConnection()**

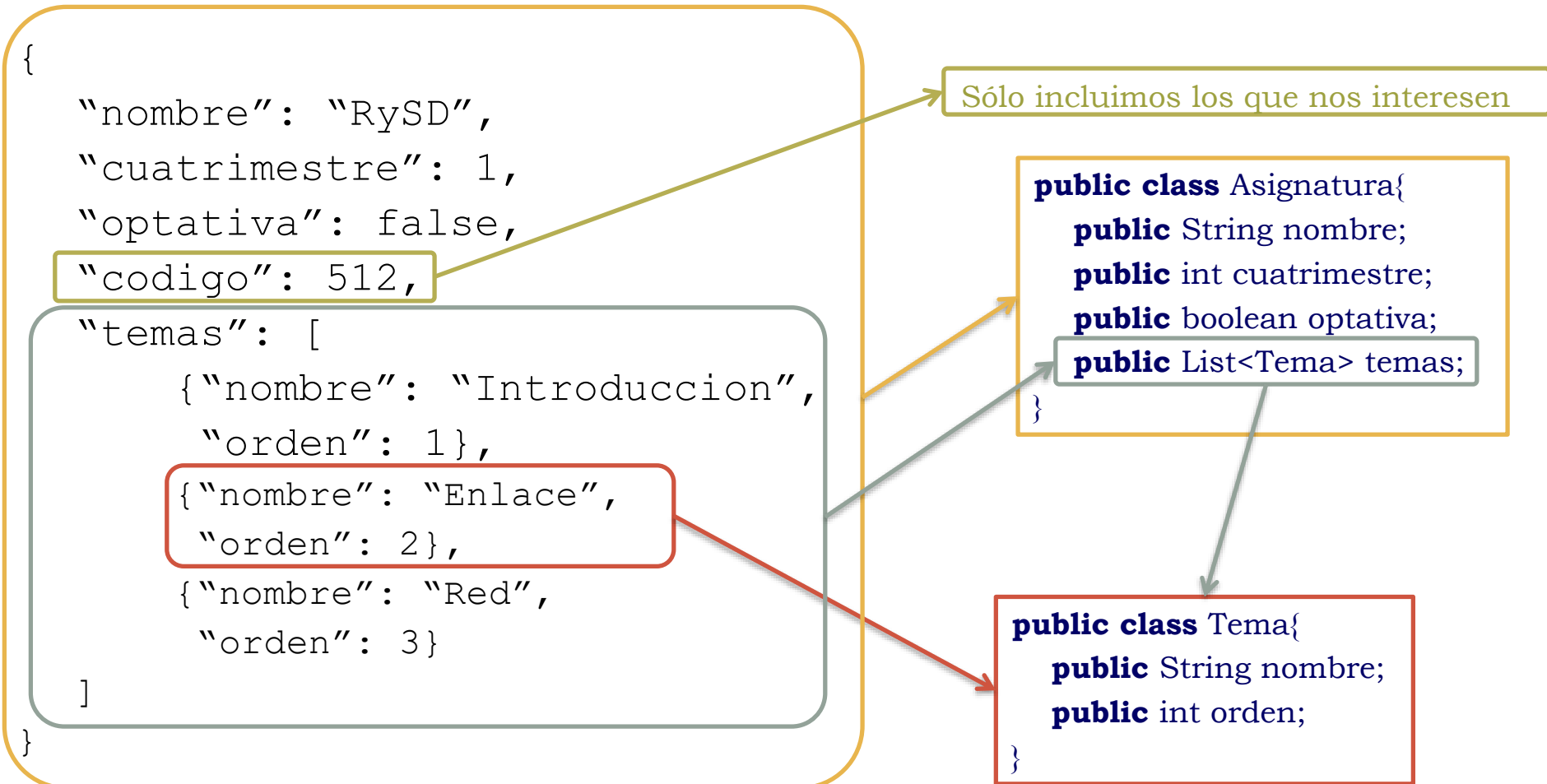
Clases Java: `URLConnection`, `HttpURLConnection`, `HttpsURLConnection`



- Permiten el manejo de conexiones http/https:
 - **`URLConnection`**: clase abstracta
 - **`HttpURLConnection`**: conexiones HTTP
 - **`HttpsURLConnection`**: conexiones HTTP seguras (HTTPS)
- Métodos más importantes:
 - Petición:
 - **`setMethod(String m)`**: establece el método (GET, POST, ...)
 - **`setRequestProperty(String name, String value)`**: Añade a la cabecera "name: value"
 - **`setDoOutput(boolean body)`**: Indica si vamos a añadir cuerpo o no a la petición
 - **`getOutputStream()`**: obtiene el flujo de datos para escribir el cuerpo
 - Respuesta:
 - **`getResponseCode()/getResponseMessage()`**: Código/mensaje de la respuesta.
 - **`getHeaderField(String name)`**: Devuelve el valor de la cabecera correspondiente a name
 - **`getInputStream()`**: obtiene un flujo para leer el cuerpo de la respuesta
 - Conexiones:
 - **`connect()/disconnect()`**: inicio/acaba la conexión

Analizando la respuesta: JSON (análisis con Gson)

- JavaScript Object Notation (JSON) es un formato ligero de intercambio de datos procedente originalmente del mundo de JavaScript
- **Gson**: biblioteca java que convierte json ↔ objetos java



Analizando la respuesta: JSON (análisis con Gson)

- Constructor:
 - **Gson()**. Ejemplo:

```
Gson g = new Gson();
```

- Convertir de JSON a objeto Java:
 - **T fromJson(JsonReader reader, Type typeOfT)**. Ejemplo:

```
InputStream is = connection.getInputStream();  
Asignatura a = g.fromJson(new InputStreamReader(is),  
                           Asignatura.class);
```

- Convertir de objeto Java a JSON:
 - **String toJson(Object src)**. Ejemplo:

```
Asignatura a = ...  
...  
String json = g.toJson(a);
```


Analizando la respuesta: JSON (análisis con Gson)

```
[{
  "nombre": "RySD",
  ...
},
{
  "nombre: "Sistemas Operativos",
  ...
},
...
]
```

Listado de
Asignaturas

- Con objetos abstractos (ej. List):

```
InputStream is = connection.getInputStream();
Type ListAsig = new TypeToken<List<Asignatura>>() {}.getType();
List<Asignatura> a = g.fromJson(new InputStreamReader(is), ListAsig);
String s = g.toJson(a, ListAsig);
```

- Con arrays:

```
InputStream is = connection.getInputStream();
Asignatura[] a = g.fromJson(new InputStreamReader(is),
                           Asignatura[].class);
String s = g.toJson(a, Asignatura[]);
```

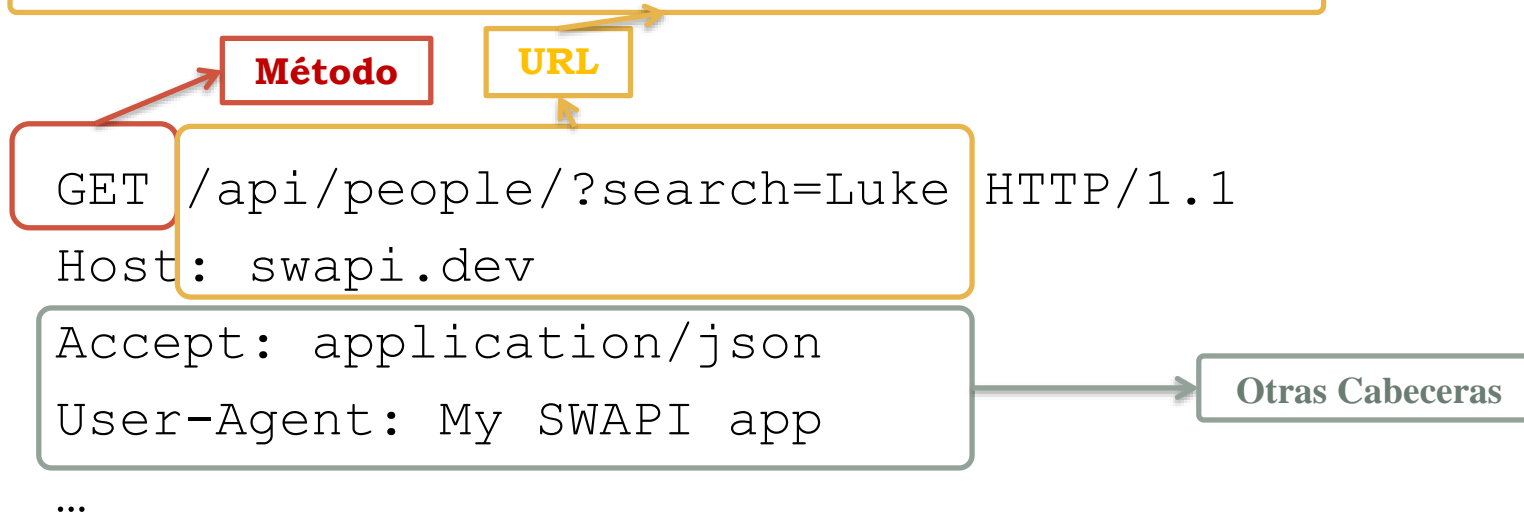
API REST SWAPI: Introducción

- SWAPI dispone un API REST público para poder hacer aplicaciones: <https://swapi.dev/documentation>
- Requiere conexiones cifradas (**https**)
- NO requiere autorización/autenticación
- Limita la cantidad de peticiones:
<https://swapi.dev/documentation#rate>

API REST SWAPI: Petición

- Ejemplo: Buscar personaje que tengan el nombre “Luke”:

<https://swapi.dev/api/people/?search=Luke>



- API:

- Planetas: <https://swapi.dev/api/planets/>
- Razas: <https://swapi.dev/api/species/>
- Películas: <https://swapi.dev/api/films/>
- Naves: <https://swapi.dev/api/starship/>

Documentación = <https://swapi.dev/api/planets/documentation>

API REST SWAPI: Respuesta

- Ejemplo de respuesta a una búsqueda similar a la mostrada antes:

