Práctica 6

Análisis con Wireshark del protocolo HTTP (versión 1.1). Creación de aplicaciones con acceso a datos distribuidos usando el protocolo HTTP en Java.

Tarea 1: Creación de un cliente básico de SWAPI en Java

Paso 1: API de Java para HTTP

Java posee una serie de clases que permiten establecer una conexión con un servidor http sin utilizar sockets manualmente. Estas clases ofrecen una API de más alto nivel para trabajar con el protocolo http. En esta práctica utilizaremos esta API para acceder al servicio Web REST. Las clases principales de esta API son: `URL`, `URLConnection`, `HttpURLConnection` y `HttpsURLConnection`.

Para utilizar estas clases, en primer lugar, debemos crear un objeto de clase `URL` que represente la URL a la que nos queremos conectar. Esto se puede hacer simplemente pasando una cadena de caracteres con la URL al constructor de la clase `URL`. Por ejemplo:

```
URL servicio = new URL("http://www.example.com/endpoint1/");
```

Una vez que tenemos la URL, debemos "abrir" una conexión asociada a ella y configurarla:

```
HttpsURLConnection connection = (HttpsURLConnection) servicio.openConnection();
connection.setRequestProperty("Accept", "application/json");
connection.setRequestProperty("User-Agent", "My simple application");
connection.setRequestMethod("GET");
```

El método **`setRequestProperty`** permite añadir cabeceras a la petición. **`setRequestMethod`** indica el método a utilizar. En el caso de que estemos utilizando la versión cifrada de HTTP (HTTPS) en vez de usar `HttpURLConnection` usaremos `HttpsURLConnection` en la conexión.

Cuando tengamos la conexión configurada, podemos conectarnos al servidor usando cualquier método que consulte el resultado. En particular, se pueden usar los métodos **`getResponseCode`**, que devuelve el código numérico de respuesta y **`getInputStream`**, que devuelve un `InputStream` para leer el contenido del cuerpo de la respuesta.

Paso 2: Biblioteca GSON

Usaremos la biblioteca denominada GSON (https://code.google.com/p/google-gson/) para serializar objetos Java en formato JSON y viceversa. Si queremos serializar un objeto Java a JSON, solo tenemos que invocar el método **`toJson`** de un objeto de la clase GSON (esa clase tiene un constructor sin argumento). Obtendremos un `String` que contiene el documento JSON. También podemos realizar el paso contrario, convirtiendo un documento JSON a un objeto Java. Esto se puede hacer invocando el método **`fromJson`**. Este método está sobrecargado, existiendo versiones que reciben un **`String`** con el documento JSON o un `Reader` (más conveniente para nuestros propósitos). Como segundo argumento debemos pasar la clase del objeto que queremos que cree. Esta clase debe tener atributos con los mismos nombres que los que tiene el objeto JSON serializado. Junto con este enunciado se proporcionan las clases para tratar con las respuestas JSON. Por ejemplo, la clase `People` representa a un personaje de la aplicación y tienen atributos que coinciden con los de los documentos JSON descargados desde SWAPI. Un ejemplo de deserialización usando esa clase es:

```
People u = gson.fromJson(new InputStreamReader(connection.getInputStream()), People.class);
```

Paso 3: Acceso a SWAPI

SWAPI (https://swapi.dev/) es una fuente de datos sobre Star Wars, utilizado por fans de las películas y en docencia. Ofrece información sobre personajes, planetas, razas, vehículos, naves de las 7 primeras películas oficiales. Ofrece una API REST (https://swapi.dev/documentation) que permite a los desarrolladores implementar aplicaciones que accedan a los servicios de SWAPI. En esta tarea vamos a acceder a varios servicios de esta API REST que requiere que el intercambio de información esté cifrado y para ello se usa https (HTTP sobre SSL). Además de este mecanismo de encriptado, no requiere ningún tipo de autenticación. Esto simplifica el uso, pero por otro lado al no tener forma de identificar al usuario, pone límite de uso por IP (10000 consultas por día).

Para acceder a la información facilitada por el API, solo hay que acceder al recurso oportuno (http://swapi.dev/api/**{recurso}**/ donde **{recurso}** puede ser **films**, **planets**, **people**, **species**, **starships** y **vehicles**) y nos devolverá como resultado un fichero JSON con la información solicitada. Dentro del JSON suelen venir URL que relacionan el recurso con otros. Por ejemplo, cuando obtenemos los datos de una persona viene la URL al recurso que representa el planeta donde nació, las URL de las películas donde sale, la URL a su especie, las URLs de las naves que ha tripulado… Cada recurso puede consultarse de tres formas diferentes:

- **[A]** http://swapi.dev/api/**{recurso}**/ Nos devuelve cuántos elementos de ese recurso hay e información sobre ellos.
- **[B]** http://swapi.dev/api/**{recurso}**/**{id}**/ Devuelve un elemento concreto (el solicitado por su identificador que es un número natural).
- **[C]** http://swapi.dev/api/**{recurso}**/**?search={nombre}** <Devuelve un conjunto de elementos cuyo nombre se ajusta con el nombre parcial indicado como parámetro.

Paso 4: Programación de un cliente básico para SWAPI

Vamos a hacer un cliente que consuma la información facilitada por SWAPI. En concreto vamos a hacer un pequeño juego de tipo trivial usando la información obtenida. La mayoría del código ya se ofrece en el código inicial (tiene comentarios para facilitar su entendimiento) y solo se solicita completar las llamadas al API de SWAPI usando HTTP para la obtención de la información.

En el campus virtual puede encontrar un código inicial para la práctica. Las clases facilitadas son:

- Clase **`es.uma.rysd.app.Main`**: incluye el programa para ejecutar la aplicación y las funciones que crean las preguntas.
- Clase **`es.uma.rysd.app.SWClient`**: encapsula la conexión con SWAPI y la obtención de datos. Esta es la clase en la que principalmente debe hacer cambios.
- Clases **`es.uma.rysd.entities.*`**: representa la deserialización de los datos recibidos en formato JSON. Aunque no requiere su modificación, puede cambiarlos si quiere para añadir más atributos incluidos en el JSON recibido o para hacer una implementación más detallada del método **`toString`**. Por completitud, se ofrecen las clases para todos los recursos que facilita el API, aunque algunos no se utilicen.

Especificación del cliente de SWAPI:

- Descargue el código e impórtelo en eclipse.
- La clase `Main`, se llaman de forma secuencial a la creación de las preguntas. Este proceso lo repite las veces que desee el usuario. Algunas de las llamadas están comentadas para facilitar la prueba parcial. En ese mismo fichero dispone del código para las preguntas. Si está en la universidad debe descomentar las primeras líneas del método **`main`** para activar el uso del proxy.
- Complete el código de la clase `SWClient` siguiendo las pautas de los siguientes puntos.
- Rellene el nombre de su aplicación.
- Como primer paso obtendremos cuantos elementos hay de un recurso implementando el método **`countResources`** de la siguiente forma:
  - Realizar una petición a la URL https://swapi.dev/api/**<recurso>**/ donde **<recurso>** es el recurso que queremos analizar y recibiremos como parámetro.
  - Debe incluir la cabecera **User-Agent** con el nombre de su aplicación (atributo `app_name`) seguido de un guión y al atributo `year`.
  - Debe incluir la cabecera **Accept** con el valor **application/json**.
  - La petición debe ser de tipo GET.
  - Compruebe que el código recibido es correcto (2XX). Si en este u otro método, al comprobar el código no da correcto devuelva 0 o null atendiendo al tipo de salida del método.
  - Deserialice el JSON recibido en un objeto de la clase `ResourceCountResult` usando GSON.
- Devuelva la cantidad de elementos que tiene dicho recursos. Como siguiente paso obtendremos los datos básicos del personaje, haciendo una primera versión del método **`getPerson`** de la siguiente forma:
  - En este caso, directamente nos pasan la URL a la que debemos consultar y los pasos son similares al método previo.
  - Solo debe tener en cuenta que ahora debe deserializar el objeto recibido usando la clase `Person`.
  - En esta primera versión no hace falta obtener los datos del planeta.
- Ya puede probar el código y ver si todo funciona de forma adecuado.
- Ahora obtendremos los datos de un planeta (**`getWorld`**) de forma similar a como obtuvo los datos del personaje (clase para deserializar: `World`).

- Modifique **`getPerson`** para que incluya información del planeta natal del personaje usando el método previo. Para ello rellene el campo **`homeplanet`** con los datos del planeta que se detallan en la URL del campo **`homeworld`**.
- En el **`main`** descomente la llamada al método **`whoBornIn2`** y pruébelo.
- Ahora buscaremos un personaje mediante su nombre usando el método **`searchPersonByName`** de la siguiente forma:
  - Realizar una petición a la URL https://swapi.dev/api/people/?search=**<nombre>** donde **<nombre>** es el personaje a buscar. El nombre debe estar codificado de forma adecuado para ser pasado como URL (use `URLEncoder.encode(nombre, "utf-8")` para ello).
  - Debe incluir la cabecera **User-Agent** con el nombre de su aplicación (atributo `app_name`) seguido de un guión y al atributo `year`.
  - Debe incluir la cabecera **Accept** con el valor **application/json**.
  - La petición debe ser de tipo GET.
  - Compruebe que el código recibido es correcto (2XX).
  - Deserialice el JSON recibido en un objeto de la clase `QueryResponse` usando GSON.
  - Devuelva los datos iniciales del personaje (devuelva siempre el primero de la lista).
  - Igual que en **`getPerson`** incluya los datos del planeta natal.
- En el **`main`** descomente la llamada al método **`whoBornIn1`** y pruébelo.
- Finalmente, implemente una pregunta similar a **`tallest`** y a **`whoBornIn2`** pero sobre otros recursos (se recomienda duplicar las preguntas en el **`main`** y luego adaptarlas a los nuevos recursos, así como crear los métodos **`getX`** para el recurso **`x`** utilizado dentro de la clase `SWClient`). Algunas ideas:
  - Nave o vehículo más caro o más grande.
  - Quién sale en la película X
  - Quién pilota la nave/vehículo X
  - Quién es de la raza X
  - En qué película sale el planeta X
  - … (sea creativo)

Tarea 2: Análisis de tráfico HTTP/1.1 generado por la aplicación

Paso 1: Captura de Tráfico HTTP/1.1 (p6e1-3.pcapng)

Como usamos tráfico encriptado (usando TLS), para poder analizarlo necesitamos las claves privadas generadas de forma temporal para la comunicación. Para conseguirlas, usaremos el módulo *jSSLKeyLog.jar* (incluido en el código facilitado). En los parámetros de la Java VM[1] debemos indicar "**-javaagent:jSSLKeyLog.jar=sslkey-app.log**". Eso generará el fichero **sslkey-app.log** que podemos incorporar a Wireshark (Edit → Preferences… → Protocols → TLS → (Pre)-Master-Secret log filename) para descifrar las comunicaciones.

Arranque Wireshark y empiece a capturar tráfico. Ejecute su programa con la opción anterior y complete de forma correcta su ejecución**.** Tras finalizar, pare la captura y guárdela con el nombre **p6e1-3.pcapng.**

Comprobación de que son correctas las trazas:

- Incorpore el fichero de claves a Wireshark.
- Filtre con **http.host == swapi.dev**
- Deben aparecer peticiones GET típicas de HTTP

Paso 2: Preguntas sobre el tráfico HTTP/1.1

Para mostrar solo el tráfico generado por la aplicación use el siguiente filtro: `http && ip.addr == IP_swapi.dev` (reemplace `IP_swapi.dev` por la IP del API). En todos los ejercicios recuerde en tomar capturas de pantalla e indicar al inicio del ejercicio qué número de trama o números de tramas ha analizado.

Ejercicio 1. ¿Cuál es el puerto utilizado por el servidor? ¿Es el normal de HTTP (80)? ¿Por qué?

Ejercicio 2. Observe el número de conexiones realizadas. ¿Cuántas hace? ¿Usa una conexión permanente (en la misma conexión hace varias peticiones) o no permanente (solo realiza una por conexión)? En caso de ser permanente, ¿qué cabecera de la petición indica que queremos que sea permanente?

---

[1] En eclipse: `Run Configurations` => Pestaña `Arguments` => Cuadro de texto: `VM arguments`
En IntelliJ: `Edit Configurations` => Cuadro de texto: `VM options`

Ejercicio 3. Describa el significado de las cabeceras de una petición y una respuesta (sin incluir las `X-*`).

Notas sobre la memoria

- Dicha memoria debe constar de una portada donde se indique la práctica que incluye la memoria, así como todos los datos del alumno.
- Esta memoria debe entregarse junto a la de la práctica 7.
- Para evaluar la parte de código se pide realizar un **vídeo donde el alumno explique el código desarrollado**. Este vídeo debe durar a lo sumo **5 minutos** (cualquier vídeo de mayor longitud no se evaluará). En el vídeo el alumno debe mostrar el código e ir explicando las partes más importantes relacionadas con las comunicaciones y sobre la pregunta adicional desarrollada. En el vídeo debe explicarse al menos los métodos **`getPerson`**, **`searchPersonByName`** y cualquier clase o método desarrollado para la nueva pregunta desarrollada.
- No es necesario copiar el enunciado completo de la práctica, pero sí debe copiarse el enunciado de cada ejercicio antes de indicar su respuesta. Debe utilizarse algún sistema de estilos que permita distinguir lo que es el enunciado de lo que es la respuesta al ejercicio.
- Para **cada ejercicio** que obtenga la información de algún proceso realizado en el ordenador (traza de Wireshark, comando...) **realice una captura** (con <alt>+<impr pant> sólo capturaremos la ventana activa actual). Además de incluir la captura se deben utilizar las herramientas de dibujo del procesador de texto usado para marcar la parte donde se observa lo que pide el ejercicio. Finalmente, en el texto añada una pequeña descripción de la captura. También indique los **números de las tramas utilizadas** para contestar la pregunta.
- El formato de entrega de **memoria** de las prácticas será **PDF**.
- Cuando se entregue la memoria se adjuntarán el fichero de trazas wireshark (fichero **p6e1-3.pcapng**), el fichero de claves (**sslkey-app.log**) utilizadas en la práctica y el código desarrollado (incluya solo los **.java** modificados, al menos será el fichero: **SWClient.java** y **Main.java**)