Actividad 1. Hilos y Sockets

Arquitectura del programa

Como paso previo comenzamos diseñando la arquitectura del programa a desarrollar. De acuerdo a los requirementos del ejercicio, y simplemente leyendo el enunciado, sabemos que necesitaremos los siguientes componentes:

1. Un cliente, donde se ejecutarán las operaciones de interacción con el servidor. El cliente va a necesitar:
    - Un socket para la comunicación con el servidor.
    - Un modelo "Libro" -para representar nuestro objeto "Libro".
    - Un servicio "LibroService" que encapsulará la lógica de todas las operaciones relacionadas con el Libro.
    - Una clase main para ejecutar el programa.
2. Un servidor, donde se ejecutarán las operaciones propias del servidor y la conexión a la base de datos. El servidor necesitará:
    - Un socket para la comunicación con el cliente.
    - Un modelo "Libro" -para representar nuestro objeto "Libro".
    - Un modelo DAO, que encapsulará la lógica de todas las operaciones relacionadas con el Libro.

En circunstancias normales existiría un modelo libro para el servidor y otro distinto para el cliente -ya que servidor y cliente van a correr en distintas máquinas-. Pero como se trata de un ejercicio de clase, y para evitar duplicar el código, crearemos un paquete "model" donde la clase libro será compartida.

Los dos programas quedarían estructurados así:

```
MODEL
    |_Libro


CLIENTE
    |_ Clase main
    |_ Socket para el cliente
    |_ service
        |_ LibroService

SERVIDOR
    |_ Clase main
    |_ DAO
    |   |_ LibroDao
    |_ Models
        |_ Libro
```

Opcionalmente, sería posible crear un "buffer libros", pero dado que el concepto del progrmaa es una "Biblioteca", se da por hecho que habrá cierta persistencia en la recopilación y almacenamiento de datos, por

lo que es más apropiado trabajar con un DAO.

Desarrollo del cliente

Comenzamos estableciendo la conexión con el servidor. Para ello, necesitamos la dirección IP y el puerto al que nos vamos a conectar. En este caso, la dirección IP será "localhost" y el puerto 2000.

```
 String ipv4 = "localhost";
        int port = 2000;
        System.out.println("APLICACIÓN CLIENTE");
        System.out.println("-----------------------------------");
        Scanner scanner = new Scanner(System.in);

        try {
            /**Un InetSocketAddress es solo una combinación de una dirección IP (o
nombre de host) y un número de puerto.
             * No es una conexión en sí misma, sino una dirección que indica a qué
IP y puerto se quiere conectar o enlazar. */
            ClienteSocket clienteSocket = new ClienteSocket(ipv4, port);
            LibroService libroService = new LibroService(clienteSocket);
```

Ciertamente, nuestra clase cliente no va a hacer nada más. Ya que el resto de la lógica quedará encapsulada dentro de nuestras clases "ClienteSocekt" y "LibroService".

Clase ClienteSocket

ClienteSocket se encargará de establecer la conexión con el servidor. Para asegurar su correcto funcionamiento, crearemos 3 métodos: uno para enviar peticiones, otro para recibir respuestas y un tercer método para cerrar la conexión.

```
package CLIENTE;

import java.io.*;
import java.net.Socket;

public class ClienteSocket {
    private Socket socket;
    private BufferedReader reader;
    private PrintWriter writer;

    public ClienteSocket(String ipv4, int port) throws IOException {
        this.socket = new Socket(ipv4, port);
        this.reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        this.writer = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()), true);
```

```
    }

    public void sendMessage(String message) throws IOException {
        writer.write(message + "\n");
        writer.flush();
    }

    public String receiveMessage() throws IOException {
        return reader.readLine();
    }

    public void close() throws IOException {
        reader.close();
        writer.close();
        socket.close();
    }

}
```

Clase LibroService

A diferencia del modelo DAO, que se encarga de operaciones relacionadas con la base de datos, un servicio simplemente nos facilita la persistencia de datos y la manera de manejar el formato en el que se va a enviar y recibir la información. Nuestra clase LibroService tendrá los siguientes métodos:

- Un método para buscar un solo libro (toma como parámetros un criterio y un valor de búsqueda).
- Un método para buscar varios libros (toma como parámetros un criterio y un valor de búsqueda).
- Un método para procesar la respuesta de búsqueda de un solo libro.
- Un método para procesar la respueta de varios libros.
- Un método para añadir un libro.
- Un método para enviar peticiones de búsqueda.

Mantener estos métodos con este formato permitirá reutilizar o ampliar el código fácilmente en el futuro si, por ejemplo, se decidiesen crear nuevos parámetros de búsqueda.

Envío de peticiones

Para optimizar el envío de peticiones, se ha optado por enviar la información en formato JSON con un "header" y un "body". Para simplificar el ejercicio al máximo, el header simplemente contendrá la información clave sobre qué debe hacer el servidor con el contenido del "body".

A la hora de enviar la información, esta se convertirá en un String de texto plano.

```
    public String sendRequest(String criterio, String value) throws IOException {
        JsonObject jsonRequest = new JsonObject();
        jsonRequest.addProperty("header", "getBy" + criterio);
        jsonRequest.addProperty("body", value);
        clienteSocket.sendMessage(jsonRequest.toString());
```

```
        return clienteSocket.receiveMessage();
    }
```

Recepción de respuesta

Inmediatamente, si la conexión es exitosa, el socket debería recibir la respuesta del servidor (que igualmente vendrá en un JSON con un header y body). Para la lectura y el procesamiento de este JSON harán falta distintos métodos, dependiendo del tipo de información que se espera recibir (si es un libro, varios o un booleano). Un ejemplo de recepción quedaría así:

```
    private Libro procesarRespuestaUnLibro(String respuesta) {
        JsonObject jsonResponse =
JsonParser.parseString(respuesta).getAsJsonObject();
        JsonObject bodyResponse = jsonResponse.getAsJsonObject("body");
        JsonObject selectedBook = bodyResponse.getAsJsonObject("content");

        if (selectedBook.size() == 0) {
            return null;
        } else {
            return gson.fromJson(selectedBook, Libro.class);
        }
    }
```

Método de unión

Como punto de unión entre el envío y la recepción de la respuesta, hay distintos métodos que se encargan de "clasificar" cómo se va a producir. Estos métodos, a su vez, tendrán nombres estandarizados para que sean fácilmente reconocibles desde el cliente:

```
    public Libro findOne(String criterio, String value) throws IOException {
        String respuesta = sendRequest(criterio, value);
        return procesarRespuestaUnLibro(respuesta);
    }

    public ArrayList<Libro> findMany(String criterio, String value) throws
IOException {
        String respuesta = sendRequest(criterio, value);
        return procesarRespuestaVariosLibros(respuesta);
    }
```

Clase main del cliente

La clase main del cliente únicamente se encargará de simular una interfaz gráfica para redirigir las consultas y, como mucho, guardar en variables las respuestas del servidor para que puedan ser utilizadas posteriormente para cualquier propósito. Para ello, solo hemos usado un menú, mensajes por consola y un switch-case.

Desarrollo del Servidor

La arquitectura del servidor, al ser menos compleja (para este ejercicio) la vamos a reducir a la clase principal (Servidor) y al modelo DAO (LibroDAO). Si el servidor siguiese creciendo, sería necesario incorporar una clase para mejorar la lógica de los sockets y la conexión, y otra clase para gestionar las respuestas.

Pero por el momento, esto es lo que encontraremos dentro del servidor:

1. Un método main con un socket para establecer la conexión con un bucle while que establecerá una conexión con cada cliente que se conecte.
2. Un método para gestionar cada petición entrante, y que contará con un bucle para seguir recibiendo peticiones hasta cerrar la conexión.
3. Un método para procesar la petición y enviar la respuesta al cliente.

Al final, el método main del servidor quedaría así:

```
public static void main(String[] args) {

        String ipv4 = "localhost";
        System.out.println("APLICACIÓN DE SERVIDOR");
        System.out.println("-----------------------------------");

        try {
            ServerSocket servidor = new ServerSocket();
            InetSocketAddress direccion = new InetSocketAddress(ipv4, 2000);
            servidor.bind(direccion);
            System.out.println("Servidor creado y escuchando .... ");
            System.out.println("Dirección IP: " + direccion.getAddress());

            while (true) {
                Socket enchufeAlCliente = servidor.accept();
                System.out.println("Comunicación entrante");

                // Creamos un nuevo hilo para manejar al cliente
                new Thread(() -> manejarCliente(enchufeAlCliente)).start();
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());

        }
    }
```

Cada nueva petición entrante creará un nuevo hilo. Dentro de ese hilo, se cursarán las peticiones de forma individual:

```
public static void manejarCliente(Socket enchufeAlCliente){
        LibroDAO biblioteca = new LibroDAO();
        try {
```

```
            //ENTRADA
            InputStream entrada = enchufeAlCliente.getInputStream();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(entrada));
            String mensaje;
            //SALIDA
            OutputStream salida = enchufeAlCliente.getOutputStream();

            while ((mensaje = reader.readLine()) != null) {

                // Objeto para RECIBIR petición
                JsonObject jsonRequest =
JsonParser.parseString(mensaje).getAsJsonObject();
                String requestHeader = jsonRequest.get("header").getAsString();
                String requestBody;
                JsonElement jsonResult;

                switch (requestHeader) {
                // AQUÍ CLASIFICAMOS LA PETICIÓN
```

Para el envío de la petición, se ha creado un método que tomará como parámetros el outputStream, el header y el contenido del body. No es imprescindible como tal, pero el método retornará la respuesta (por si fuese necesario utilizarla en un futuro para algo).

```
 private static JsonObject sendResponse(String headerContent, JsonElement
bodyContent, OutputStream salida ) throws IOException {
        JsonObject jsonResponse = new JsonObject();
        JsonObject responseHeader = new JsonObject();
        JsonObject responseBody = new JsonObject();
        jsonResponse.add("header", responseHeader);
        jsonResponse.add("body", responseBody);
        responseHeader.addProperty("header", headerContent);
        responseBody.add("content", bodyContent);
        System.out.println("SALIDA DEL SERVIDOR: " + jsonResponse);
        salida.write((jsonResponse.toString() + "\n").getBytes() );
        return jsonResponse;
    }
```

Modelos DAO (synchronus) y Libro

En este punto no nos entretendremos demasiado. Es suficinete con saber que tanto el DAO como el Libro encapsulan la lógica de los objetos creados y las operaciones CRUD (aunque en este ejercicio solo se presentan operaciones de consulta y añadido).

Como detalle importante, la operación "añadir" debe quedar bloqueada en caso de que uno de los hilos acceda a ella, y solamente desbloqueará el programa cuando concluya. Para simular este "bloqueo" -y comprobar que efectivamente el programa ha quedado bloqueado- se ha decido implementar un

Threat.sleep(3000) para simular que el bloqueo funciona correctamente, es decir, nunca van a llegar varias peticiones de añadir libro simultáneamente.

```
    public synchronized JsonArray add(JsonObject bookObject) {
        JsonArray jsonBookList = new JsonArray();

        String isbn = bookObject.get("ISBN").getAsString();
        String title = bookObject.get("title").getAsString();
        String author = bookObject.get("author").getAsString();
        String prize = bookObject.get("prize").getAsString();
        System.out.println("AÑADIENDO LIBRO: " + bookObject.toString());

        Libro nuevoLibro = new Libro(isbn, title, author, prize);
        try {
            /*Simulamos un retraso de 4 segundos.
            * Si el hilo está bien sincronizado, nunca podrán salir por consola
            * simultaneamente varios mensajes, ya que el hilo está bloqueando el
proceso.*/
            Thread.sleep(4000);

            if (findByIsbn(nuevoLibro.getISBN()).size() != 0){
                System.out.println("El libro ya existe");
            } else {
                listaLibros.add(nuevoLibro);
                JsonObject libroJson =
gson.toJsonTree(nuevoLibro).getAsJsonObject();
                jsonBookList.add(libroJson);
            }

        } catch (InterruptedException e) {
            Thread.currentThread().interrupt(); // Restaura el estado de
interrupción
        } catch (Exception e){
            System.out.println("Datos incompletos, introduce un formato de libro
válido");
        }
        return jsonBookList;
    }
```

Cabe destacar que el método añadir devolverá un JsonArray que podría estar vacío -si el libro ya existe-, o contener el libro añadido. En realidad esto no es importante. Lo único que nos interesa es tener un parámetro con el que saber si se ha añadido el libro con éxito o no. Así, dentro del cliente, solo tendremos que verificar el tamaño del JsonArray. Otra forma más elegante de manejarlo habría sido con un booleano. Aunque en este caso se ha optado por el JsonArray para reutilizar los métodos que ya teníamos creados en el cliente.

Observaciones

- Dentro del código hay varios mensajes por consola para depurar y comprobar que el programa funciona correctamente. En un caso real, estos mensajes deberían ser eliminados o comentados.

- La propiedad "prize" del libro se ha declarado como String, aunque en un caso real debería ser un tipo numérico (double o float). Al no haber operaciones aritméticas, y para evitar tener que realizar un casting, se ha simplificado como String.

- En un caso real, y dentro del cliente, verificaríamos que el JSON que está llegando se corresponde exactamente con el tipo que queremos manejar. Es decir, nosotros damos por hecho que al recibir un JSON Object de verdad se trata de un Json Object, y por ello manipulamos el objeto directamente, aunque esto es algo que debemos verificar antes de ejecutar.

Resultados

A continuación se presentan los patanllazos de la ejecución del programa:

```
Introduce el título
anillos
Encontrado:
[Libro{ISBN='3C', titulo='El señor de los anillos: La
]
Menú de opciones:
1. Consultar libro por ISBN
2. Consultar libro por título
3. Consulta libro por autor
4. Añadir libro
5. Salir de la aplicación
Seleccione una opción: 3
Introduce el autor
Tolkien
Encontrado:
[Libro{ISBN='3C', titulo='El señor de los anillos: La
, Libro{ISBN='4D', titulo='El hobbit', autor='J.R.R.
]
Menú de opciones:
1. Consultar libro por ISBN
2. Consultar libro por título
3. Consulta libro por autor
4. Añadir libro
5. Salir de la aplicación
Seleccione una opción: 3
Introduce el autor
anonimo
No hay coincidencias de búsqueda
```

```
No hay coincidencias de búsqueda
Menú de opciones:
1. Consultar libro por ISBN
2. Consultar libro por título
3. Consulta libro por autor
4. Añadir libro
5. Salir de la aplicación
Seleccione una opción: 4
Introduce un ISBN
1
Introduce un título
1
Introduce un autor
1
Introduce un precio
1
Libro añadido correctamente
Menú de opciones:
1. Consultar libro por ISBN
2. Consultar libro por título
3. Consulta libro por autor
4. Añadir libro
5. Salir de la aplicación
Seleccione una opción: 1
Introduce el ISBN
1
Encontrado:
Libro{ISBN='1', titulo='1', autor='1', prize=1}
```

```
APLICACIÓN DE SERVIDOR
---------------------------------
Servidor creado y escuchando ....
Dirección IP: localhost/127.0.0.1
Comunicación entrante
SALIDA DEL SERVIDOR: {"header":{"header":"getByISBN"},"body":{"content":{"ISBN":"1A","title":"Harry Potter y la piedra filoso
SALIDA DEL SERVIDOR: {"header":{"header":"getByTitle"},"body":{"content":[{"ISBN":"3C","title":"El señor de los anillos: Las
SALIDA DEL SERVIDOR: {"header":{"header":"getByTitle"},"body":{"content":[{"ISBN":"3C","title":"El señor de los anillos: Las
SALIDA DEL SERVIDOR: {"header":{"header":"getByTitle"},"body":{"content":[]}}
AÑADIENDO LIBRO: {"ISBN":"1","title":"1","author":"1","prize":"1"}
SALIDA DEL SERVIDOR: {"header":{"header":"add"},"body":{"content":[{"ISBN":"1","title":"1","author":"1","prize":"1"}]}}
SALIDA DEL SERVIDOR: {"header":{"header":"getByISBN"},"body":{"content":{"ISBN":"1","title":"1","author":"1","prize":"1"}}}
```