Hilos y Sockets

Programación de Servicios y Procesos

Israel Serrano
Pablo Baldazo
Jorge Buendía

Programación de Servicios
y Procesos
2ºDAM

Información del documento

Nombre del documento	Hilos y Sockets
Autor del documento	Israel Serrano, Pablo Baldazo, Jorge Buendía
Versión del proyecto	v1.0
Fecha de entrega	31.10.2022
Repositorio GitHub	https://github.com/PIJ-Group/Hilos_y_Sockets.git

Distribución y autores del documento final

Nombre	Organización/Puesto
Israel Serrano	Todos los integrantes del equipo, hemos trabajado de manera conjunta durante la realización de este proyecto. Hemos dado importancia a la funcionalidad principal del programa, la cual, hemos desarrollado en conjunto. Posteriormente, cada miembro del equipo se centró en resolver distintos problemas que nos iban surgiendo durante el desarrollo del proyecto. Finalmente, pusimos en común todas las soluciones, para decidir cuál era la mejor y también conocer los errores cometidos.
Pablo Baldazo	
Jorge Buendía	

Contenido

1 Introducción

1.1 Objetivo

El objetivo de este proyecto es el de crear una aplicación que administre una serie de libros, simulando una biblioteca virtual que será gestionada a través de dos máquinas cliente-servidor que estén conectadas a través de sockets y se genere un hilo por cada cliente.

2 Requerimientos

Requerimiento 1

Se pide hacer dos programas cliente-servidor con sockets e hilos. La aplicación servidora programa consistirá en crear una aplicación que gestione una serie de libros de una biblioteca virtual, la aplicación cliente consumirá dicha aplicación servidora.

Los libros tendrán un ISBN, un título, un autor y un precio. Se encontrarán alojados en el servidor. Dicho servidor cuando arranque tendrá 5 libros preestablecidos con todos los datos rellenos. Los libros se guardarán en memoria en cualquier tipo de estructura de datos (como puede ser una lista). El servidor **deberá estar preparado para que interactúen con él varios clientes** (se deberá abrir un hilo por cada cliente).

La aplicación cliente mostrará un menú como el que sigue:

- Consultar libro por ISBN
- Consultar libro por titulo
- Salir de la aplicación

La aplicación se ejecutará hasta que el cliente decida pulsar la opción de “salir de la aplicación”.

El cliente deberá de recoger todos los datos del usuario necesarios y mandarlos al servidor en un solo envío.

Requerimiento 2

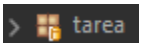
Se pide añadir otra opción que sea “Consultar libros por autor”. En este caso hay que tener en cuenta que puede haber varios libros por autor, por lo que el servidor podrá devolver una lista de libros. Se recomienda pensar en grupo el formato de envío de información.

Requerimiento 3

Se pide añadir otra opción que sea “Añadir libro”. En este caso el cliente pedirá todos los datos del libro y los enviará al servidor para que este lo guarde en el servidor. La lista en el servidor deberá estar preparada para que solo pueda añadir un libro cada hilo a la vez, si algún hilo está agregando un libro, los demás hilos deberán de esperar a que el hilo acabe.

El cliente deberá de recoger todos los datos del usuario y mandarlos al servidor en un solo envío.

3 Explicación de la actividad

Para la realización de este proyecto, hemos modularizado el trabajo a partir de cuatro grandes clases que hemos introducido dentro del paquete 

y que explicaremos a continuación:

3.1 Clase Libro

Creamos una clase para generar los libros a introducir en el servidor, tanto los que dejemos precargados en el mismo, como los que se añadan posteriormente a través del cliente.

```java
package tarea;
public class Libro {

	private String isbn;
	private String titulo;
	private String autor;
	private String precio;

	public Libro(String isbn, String titulo, String autor, String precio){

		super();
		this.isbn = isbn;
		this.titulo = titulo;
		this.autor = autor;
		this.precio = precio;

	}

	public Libro() {
		super();
	}
```

```
	public String getIsbn() {

		return isbn;
	}

	public void setIsbn(String isbn) {
		this.isbn = isbn;
	}

	public String getTitulo() {
		return titulo;
	}

	public void setTitulo(String titulo){
		this.titulo = titulo;
	}
	public String getAutor(){
		return autor;
	}

	public void setAutor(String autor){
		this.autor = autor;
	}

	public String getPrecio() {
		return precio;
	}

	public void setPrecio(String precio){
		this.precio = precio;
	}


	@Override
	public String toString() {
		String resultado = "Libro ISBN=" + isbn + ", titulo=" + titulo +
	     ,“autor=" + autor + ", precio=" + precio ;

		return resultado;
		}
```

3.2 Clase Biblioteca

Ahora procedemos a crear nuestro servidor, al que hemos llamado biblioteca, puesto que va a almacenar libros.

Empezamos declarando una constante con el puerto por el que se va a hacer la conexión, poniendo un contador de peticiones de los clientes y un ArrayList para meter los libros.

A continuación, en el método Main, imprimimos en consola un mensaje de entrada a la biblioteca, creamos el socket de dirección, cargándole el puerto definido anteriormente y sacamos un mensaje que indica que vamos a proceder a cargar los libros de la biblioteca. Introducimos un *Sleep* para simular un efecto de carga en los mismos durante un corto periodo de tiempo, y añadimos los 5 libros que vamos a tener precargados para hacer las respectivas consultas.

```
package tarea;

import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;

public class Biblioteca {

	private static final int PUERTO = 2047;
	private static int peticion = 0;
	static ArrayList<Libro>libros = new ArrayList<>();

	public static void main(String[] args) {

		System.out.println("              BIBLIOTECA              ");
		System.out.println("--------------------------------------");

		InetSocketAddress direccion = new InetSocketAddress(PUERTO);

		System.out.println("Precargando los libros de la biblioteca");

		try {

			Thread.sleep(2000);

		} catch (InterruptedException e1) {

			e1.printStackTrace();
		}
```

```
            libros.add(new Libro("9780765311771", "Elantris", "Brandon
             Sanderson", "26.50"));

            libros.add(new Libro("9788468402772", "Los muertos vivientes",
             “Robert Kirkman", "20"));

            libros.add(new Libro("9788417347291", "El imperio final",
             “Brandon Sanderson", "19.86"));

            libros.add(new Libro("8483468018", "La niebla", "Stephen King",
             “4.85"));
            libros.add(new Libro("9780007136599", "The Fellowship of The
            Ring", "J.R.R. Tolkien","253.04"));

            System.out.println("Biblioteca cargada");
```

Lo siguiente que hacemos, es crear un ServerSocket y le asignamos la dirección por la que va a enviar y recibir.
Hacemos un bucle en el que por cada petición que tengamos de conexión a nuestro servidor, se abra un Socket nuevo y se cree un hilo para ese Socket.

```
      try(ServerSocket servidor = new ServerSocket()){


            servidor.bind(direccion);

            System.out.println("Esperando petición por el puerto " +
             PUERTO);

            while(true) {

                  Socket socketCliente = servidor.accept();
                  System.out.println("\nPetición nº " + ++peticion +
                  “Recibida" + "\n" );



            new Hilos(socketCliente);

                  }

            }catch(IOException e) {

                  System.err.println("Error entrada/salida");
                  e.printStackTrace();

            }catch(Exception e) {

                  System.err.println("Error del servidor");
                  e.printStackTrace();
            }
```

Durante el desarrollo de las distintas Clases, hemos capturado diferentes errores utilizando bloques *try-catch*. Hemos tenido en cuenta, errores de **entrada/salida** en la comunicación entre usuario y biblioteca utilizando ***IOException***, así como errores generales en cada clase tales como "Errores en el servidor" o "Errores en el cliente".

También hemos gestionado otros errores más específicos que heredan de ***UnknownHostException***, cuando la dirección del servidor no ha podido ser encontrada.

3.3 Clase Hilo

Creamos la clase **Hilo**, que será la encargada de crear un hilo por cada Socket que se cree en el servidor (Biblioteca), además de implementar la lógica de funcionamiento del menú incluido en el cliente (Usuario).

Primero, para poder crear un hilo, decidimos que la clase implemente la interfaz Runnable, para lo cual tendremos que crear un nuevo objeto Thread y así poder acceder a sus métodos.

Declaramos una serie de variables, las cuales utilizaremos más adelante en la clase, destacando la variable de la clase Thread y Socket, que utilizaremos para conectar los hilos con los Socket del servidor creados y dar la capacidad a la aplicación de multiprocesamiento.

```
package tarea;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.util.ArrayList;

public class Hilos implements Runnable {

	private Thread hilo;
	private static int numeroUsuario = 0;
	private Socket socketCliente;
	private String error = "El libro no está en la Base de Datos de la
	 Biblioteca";

	public Hilos(Socket socketCliente) {

		numeroUsuario++;
		hilo = new Thread(this, "Usuario_" + numeroUsuario);
		this.socketCliente = socketCliente;
		hilo.start()
		}
```

Creamos un constructor para el hilo, al cual le pasamos como argumento el objeto Socket creado en el servidor, ya que es allí donde creamos el objeto Hilo.

Dentro del constructor, creamos el objeto Thread para poder acceder a sus métodos, y lo inicializamos, para así poder iniciar el hilo cada vez que se cree un nuevo Socket en el servidor.
A continuación, creamos un Try dentro del método run que vamos a sobrescribir, donde estará toda la lógica de la aplicación servidor.

Declaramos las variables que contendrán los objetos de los streams de entrada y salida ("entrada" y "salida") y el objeto para poder ayudarnos con la lectura del stream de entrada ("entradaBr").

Una vez creados los objetos streams, creamos un bloque while con una condición de *true/false*, en el que pasaremos a leer el stream de entrada, y dependiendo del resultado, puede realizar varias acciones:

1- Cerrar la conexión del hilo/socket y devolver el resultado al Cliente para que también cierre su aplicación.

2- Realizar las funciones descritas para cada tipo de valor del menú del cliente y devolver el resultado.

En el caso de la acción 1, en el bloque "IF" comparamos la entrada del stream con el valor 5 (el que hemos tomado como instrucción de cierre), y si es así, cambiamos la condición para que salga del bucle y cierre el Socket del servidor

```
try {

	entrada = new InputStreamReader(socketCliente.getInputStream());
	salida = new PrintStream(socketCliente.getOutputStream());
	entradaBr = new BufferedReader(entrada);

		String text;
		boolean control = true;

		while(control) {

			text = entradaBr.readLine();

			if(text.trim().equalsIgnoreCase("5")) {

				salida.println("5");
```

```
                        System.out.println("\nConexión cerrada por el " +
                         hilo.getName());

                        salida.println("Has salido de la aplicación"
                         + "\n" + "Que tenga un buen día");


                        control = false;
                    }else{
                        String info;
                        Libro info1;
                        String libroCompleto;
                        String[] libroCompuestoSeparado;
```

En el caso de la acción 2, dentro del bloque “ELSE”, creamos un bloque “TRY-CATCH” donde recogemos la lógica del menú, en el cual integramos un Switch con el que poder manejar más cómodamente cada supuesto.

En cada caso, en la clase usuario se hace una programación en “espejo”, la cual recoge los valores que enviamos desde el servidor, nos responde al servidor, y desde aquí lo tratamos para devolver la función que desee.

En los supuestos 1, 2 y 3, se ha decidido por una interacción línea a línea entre cliente y servidor, para poder dejar más claro al cliente qué acción tiene que realizar en ese momento.

En el supuesto 4, sin embargo, se ha decidido que todo el mensaje a tratar sea incluido en una sola línea, haciéndole saber al cliente cuál es el método con el que tiene que enviar la información. El formato elegido es la separación de cada apartado del libro con un guion, que luego separaremos mediante el método Split e incluyendo cada parte separada en el objeto Libro.

```
Try{
switch (text) {

      case "1":

       System.out.println("El " +  hilo.getName() + " ha solicitado la
       opción  "+text);
       salida.println("Introduzca el ISBN");
       String isbn = entradaBr.readLine();
       System.out.println("El " +  hilo.getName() + " solicita información
       a través del ISBN: " + isbn);
       info = libroIsbn(isbn);
       System.out.println("la info solicitada por el " +  hilo.getName() +
       "es: \n" +info);
       salida.println(info);
       break;
```

```
 case "2":

 System.out.println("El " +  hilo.getName() + " ha solicitado la
 opción" +text);
 salida.println("Introduzca el Título del libro");
 String titulo =      entradaBr.readLine();
 System.out.println("El " +  hilo.getName() + " solicita información a
 través del título: " + titulo);
 info = libroTitulo(titulo);
 System.out.println("la info solicitada por el " + hilo.getName() + "es:
 \n" +info);
 salida.println(info);
 break;

case "3":

System.out.println("El " +  hilo.getName() + " ha solicitado la opción"
 +text);
salida.println("Introduzca el Autor a consultar");
String autor = entradaBr.readLine();
System.out.println("El " +  hilo.getName() + " solicita información a
través  del autor: " + autor);
info = libroAutor(autor);
System.out.println("la info solicitada por el " +  hilo.getName() + "es:
 \n" +info);
salida.println(info);
break;
```

Como se puede observar, las opciones 1, 2 y 3 tienen la misma funcionalidad y estructura, con la diferencia de que, en cada caso, se llamará a una función distinta que tratará con el dato de entrada de forma diferente para devolver los datos de la opción elegida.

En el caso 1, pediremos que introduzca el ISBN y devolveremos el libro.

En el caso 2, pediremos que introduzca el título del libro y devolveremos el libro.

En el caso 3, pediremos que introduzca el autor del libro y devolveremos el libro.

```
    case "4":
    System.out.println("El " +  hilo.getName() + " ha solicitado la opción"
     +text "\n");
    salida.println("Introduzca los datos según el formato especificado:
    ISBN-Título-Autor-Precio");
    libroCompleto = entradaBr.readLine();
    libroCompuestoSeparado = libroCompleto.split("-");
    String isbn1 = libroCompuestoSeparado[0];
    String titulo1= libroCompuestoSeparado[1];
    String autor1 = libroCompuestoSeparado[2];
    String precio1 = libroCompuestoSeparado[3];
    info1 = añadirLibros(isbn1, titulo1, autor1, precio1);
    salida.println("Libro añadido correctamente");
    salida.println(info1);
    System.out.println("El libro añadido por el " +  hilo.getName() + " es:
    \n" +info1);
    break;

default:
     salida.println("Opción errónea"
     break;
}

}catch(Exception e){
System.err.println("Opción errónea");
e.printStackTrace();
}
  }
    }
socketCliente.close();

} catch (IOException e) {

System.err.println("Error entrada/salida");
e.printStackTrace();

} catch (Exception e) {

System.err.println("Error del hilo");
e.printStackTrace();
}
```

Para el caso 4, como hemos adelantado anteriormente, la estructura y funcionamiento es diferente, ya que, en vez de recibir solo un texto con el valor a consultar, recibiremos una cadena que contiene todos los valores a ingresar del libro para su creación.

En nuestro caso, hemos optado por la separación entre guiones, separándolos luego con el método Split, y creando cuatro variables que contengan cada valor separado en el orden correcto.

Una vez separado, creamos el libro con las cuatro variables que hemos creado.

Además, hemos creado un apartado "*default*" por si el cliente elige una opción mayor a 5, en ese caso devolvemos el mensaje de "opción errónea".

En el bloque **CATCH** recogemos las excepciones **IOException** y una excepción general, incluyendo un mensaje de error mediante el uso de System.err.println.

```
public String libroIsbn(String isbn){

		for(Libro libro : Biblioteca.libros) {
			if(libro.getIsbn().equalsIgnoreCase(isbn))
				return libro.toString() + "\n";
		}

		return error + "\n";
	}

	public String libroTitulo(String titulo){

		for(Libro libro : Biblioteca.libros) {
			if(libro.getTitulo().equalsIgnoreCase(titulo))
				return libro.toString() + "\n";

		}

		return error + "\n";
	}


	public String libroAutor(String autor){

		ArrayList<Libro>lista2 = new ArrayList<Libro>();
		for(Libro libro : Biblioteca.libros) {
			if(libro.getAutor().equalsIgnoreCase(autor))
				lista2.add(libro);

		}

		return lista2.toString() + "\n";

	}
```

```
  public synchronized Libro añadirLibros(String isbn1, String titulo1,
 String autor1, String precio1) {

            int lastIdx = 0;

    Biblioteca.libros.add(new Libro(isbn1, titulo1, autor1, precio1));
            lastIdx = Biblioteca.libros.size() - 1;

       return Biblioteca.libros.get(lastIdx);

     }
}
```

Por último, pasamos a detallar las funciones de cada apartado del libro, las cuales hemos separado del cuerpo principal de la clase para mejor claridad en el código.

- **Función libroISBN**: pasándole como argumento el texto de ISBN solicitado al cliente, recorre el *ArrayList* creado como base de datos de los libros y creamos un *IF* en el que comparamos el argumento con el ISBN de cada libro, y si es igual, lo muestra.

- **Función libroTitulo**: pasándole como argumento el texto de título solicitado al cliente, recorre el *ArrayList* creado como base de datos de los libros y creamos un IF en el que comparamos el argumento con el título de cada libro, y si es igual, lo muestra.

- **Función libroAutor**: pasándole como argumento el texto de autor solicitado al cliente, recorre el *Arraylist* creado como base de datos de los libros y creamos un IF en el que comparamos el argumento con el autor de cada libro, y si es igual, lo muestra.

- **Función añadirLibros**: pasándole como argumento las variables creadas con cada sección del texto solicitado al cliente, añadimos al *ArrayList* creado como base de datos de los libros, un nuevo objeto libro, y se lo mostramos al cliente para que vea que se ha creado de la manera correcta.
  En este caso, debido a los requerimientos de la práctica, este método será sincronizado (**synchronized**), para que mientras otro cliente esté añadiendo un libro, ningún otro pueda realizarlo a la vez.

3.4 Clase Usuario

En este Clase, el usuario va a interaccionar con la biblioteca a través de un menú, pudiendo consultar libros de diferentes maneras, añadiendo otros nuevos o cerrando las comunicaciones con la misma.

```
package tarea;
import java.io.BufferedReader;
public class Usuario {

	private static final int PUERTO = 2047;
	private static final String IP_SERVER = "localhost";
	public static Scanner sc = new Scanner(System.in);

	public static void main(String[] args) {

	System.out.println("        APLICACIÓN USUARIO         ");
	System.out.println("-----------------------------------");

	InputStreamReader entrada;
	PrintStream salida;
	BufferedReader entradaBr;
	InetSocketAddress direccionServer =
	 new InetSocketAddress(IP_SERVER,PUERTO);

	try{

	System.out.println("Esperando la respuesta del servidor");
	Socket socketServidor = new Socket();
	socketServidor.connect(direccionServer);
	System.out.println("Conexión establecida con la dirección
	 " + IP_SERVER + " por el puerto " + PUERTO + "\n");

	entrada = new InputStreamReader(socketServidor.getInputStream());
	salida = new PrintStream(socketServidor.getOutputStream());
	entradaBr = new BufferedReader(entrada);
```

Primero declaramos la IP y el puerto al que nos vamos a conectar, acto seguido creamos los objetos que van a definir el canal de entrada del usuario por el cual recibirá la información que nos manda el servidor (*InputStreamReader*) y el canal de salida del usuario por el cual vamos a mandar la información al servidor (*PrintStream*).

También definimos el objeto socketServidor, que a través de la clase *Socket* nos permitirá conectarnos con el servidor, además del objeto direccionServer que nos permitirá encapsular la ip y el puerto al que nos vamos a conectar.

```
      String text;
      boolean control = true;
      String opcion;
do {
System.out.println("---Bienvenido a la biblioteca virtual---");
      System.out.println("Escoja una de las siguientes opciones\n");

            opcion = menu();
            salida.println(opcion);

            System.out.println("Esperando respuesta del servidor");
            Thread.sleep(2000);
            String datosObtenidos = entradaBr.readLine();


            if("5".equalsIgnoreCase(datosObtenidos)) {
                    control = false;
                    System.out.println("Conexión terminada");

                    } else {

                          System.out.println(datosObtenidos);
                          text = sc.nextLine();
                          salida.println(text);
                          String datosObtenidos2 = entradaBr.readLine();
                          System.out.println(datosObtenidos2 + "\n");
                          String datosObtenidos3 = entradaBr.readLine();
                          System.out.println(datosObtenidos3 + "\n");
                    }


                  }while(control);

                  socketServidor.close();
```

Introducimos un bucle *do while*, que va a permitir sacar indefinidamente un menú de opciones, permitiendo introducir al usuario la opción que desee, hasta que dicho usuario envíe el número “5” al servidor. Lo que provocará que salga del bucle al cambiar la variable booleana “control”, de *True* a *False*.

En caso de no elegir la opción número “5”, el servidor mandará una respuesta con los datos solicitados al usuario, dicha respuesta será gestionada por el objeto entradaBr, que deriva de la clase *BufferedReader*, lo que nos permite leer un flujo de entrada de caracteres almacenado en búfer que proporciona una lectura eficiente de caracteres, matrices y líneas.

```
}catch (UnknownHostException e){

  System.err.println("Servidor " + IP_SERVER + " no encontrado");
  e.printStackTrace();

  }catch (IOException e) {

  System.err.println("Error entrada/salida");
  e.printStackTrace();

  }catch (Exception e) {

  System.err.println("Error de cliente");
  e.printStackTrace();

      }

  }

  public static String menu() {

      System.out.println("1. Consultar libro por ISBN");
      System.out.println("2. Consultar libro por título");
      System.out.println("3. Consultar libro por autor");
      System.out.println("4. Añadir libro");
      System.out.println("5. Salir de la aplicación");
      String option = sc.nextLine();
      System.out.println("Has elegido la opción: " + option);
      return option;
```

Finalmente volvemos a gestionar todos los errores que se puedan producir durante la ejecución del programa y hacemos una pequeña mención al método que genera el menú, donde establecemos las opciones que tiene el usuario, recogemos dicha opción y la retornamos para poder enviarla al servidor, como hemos visto anteriormente.

4 Resultados

En este apartado vamos a mostrar la funcionalidad de nuestro servidor – cliente.

- Lo primero que hacemos es arrancar la biblioteca (el servidor) obteniendo el siguiente mensaje:

```
             BIBLIOTECA
-------------------------------------
Precargando los libros de la biblioteca
Biblioteca cargada
Esperando petición por el puerto 2047
```

- A continuación, arrancamos un cliente, que en nuestro caso hemos llamado Usuario, dejándonos en la consola del servidor el mensaje de petición de conexión, y que se conecta el Usuario_1

```
Petición nº 1 recibida

Estableciendo conexión con: Usuario_1
```

Mientras tanto, en la consola del cliente nos indica el puerto por el que conectamos con el servidor, y nos da el mensaje de bienvenida con el menú para hacer las diferentes consultas.

```
       APLICACIÓN USUARIO
---------------------------------
Esperando la respuesta del servidor
Conexión establecida con la dirección localhost por el puerto 2047

---Bienvenido a la biblioteca virtual---
Escoja una de las siguientes opciones

1. Consultar libro por ISBN
2. Consultar libro por título
3. Consultar libro por autor
4. Añadir libro
5. Salir de la aplicación
```

- Ahora vamos a probar las diferentes funcionalidades:

    - **Consultar libro por ISBN**

      Hacemos nuestra primera consulta, pidiendo que nos saque un libro por su código **ISBN**, en este caso el 9788468402772

```
1
Has elegido la opción: 1
Esperando respuesta del servidor
Introduzca el ISBN
9780765311771
```

En la consola del servidor obtenemos que el **Usuario_1** solicitó la opción 1

```
El Usuario_1 ha solicitado la opción 1
```

En la consola del cliente nos da el resultado a la consulta y nos vuelve a sacar el menú para seguir navegando por las opciones.

```
Libro ISBN=9788468402772, titulo=Los muertos vivientes, autor=Robert Kirkman,
precio=20

Escoja una de las siguientes opciones

1. Consultar libro por ISBN
2. Consultar libro por título
3. Consultar libro por autor
4. Añadir libro
5. Salir de la aplicación
```

A su vez en la consola del servidor nos va dando trazas de que es lo que va consultando el **Usuario_1** y la información que se le envía.

```
El Usuario_1 solicita información a través del ISBN: 9788468402772
la info solicitada por el Usuario_1 es:
Libro ISBN=9788468402772, titulo=Los muertos vivientes, autor=Robert Kirkman,
precio=20
```

En el caso de introducir un ISBN incorrecto o que no esté en la base de datos de la biblioteca, nos dará un mensaje al respecto.

```
1
Has elegido la opción: 1
Esperando respuesta del servidor
Introduzca el ISBN
6456546545654
El libro no está en la Base de Datos de la Biblioteca
```

- **Consultar libro por Título**

A continuación, mostraremos la funcionalidad de la **consulta por título**, realizada por el mismo Usuario.
En la consola del **Usuario_1** obtendremos lo siguiente:

```
2
Has elegido la opción: 2
Esperando respuesta del servidor
Introduzca el Título del libro
La niebla
Libro ISBN=8483468018, titulo=La niebla, autor=Stephen King, precio=4.85
```

Y en la consola de la Biblioteca (el servidor) obtendremos la siguiente información:

```
El Usuario_1 ha solicitado la opción 2
El Usuario_1 solicita información a través del título: La niebla
la info solicitada por el Usuario_1 es:
Libro ISBN=8483468018, titulo=La niebla, autor=Stephen King, precio=4.85
```

Al igual que en la consulta por autor, si se introduce un autor del que no hay ningún libro en la BDD sale un mensaje informándolo.

```
2
Has elegido la opción: 2
Esperando respuesta del servidor
Introduzca el Título del libro
El trono de huesos de dragón
El libro no está en la Base de Datos de la Biblioteca
```

- **Consultar libro por Autor:**

La consulta por el **autor** la mostraremos de la misma manera que la anterior, para no redundar en lo mismo.

La consola del **Usuario_1** imprimiría lo siguiente, mostrando los dos libros que hay de ese mismo autor.

```
3
Has elegido la opción: 3
Esperando respuesta del servidor
Introduzca el Autor a consultar
Brandon Sanderson
[Libro ISBN=9780765311771, titulo=Elantris, autor=Brandon Sanderson,
precio=26.50,
Libro ISBN=9788417347291, titulo=El imperio final, autor=Brandon Sanderson
precio=19.86]
```

Y la consola de la Biblioteca hace lo propio:

```
El Usuario_1 ha solicitado la opción 3
El Usuario_1 solicita información a través del autor: Brandon Sanderson
la info solicitada por el Usuario_1 es:
[Libro ISBN=9780765311771, titulo=Elantris, autor=Brandon Sanderson,
precio=26.50,
Libro ISBN=9788417347291, titulo=El imperio final, autor=Brandon Sanderson,
precio=19.86]
```

Como en los dos casos anteriores, el Usuario, también puede ingresar un dato que no se encuentre en la BBDD. En este caso saca por consola un libro vacío, puesto que no hay datos de ese autor.

```
3
Has elegido la opción: 3
Esperando respuesta del servidor
Introduzca el Autor a consultar
Sir Arthur Conan Doyle
[]
```

- **Añadir libro:**

Vamos a ver la funcionalidad de la opción 4, la de **añadir libro**. En este caso, se le pide al **Usuario_1** que introduzca los datos con un formato específico, para recoger toda la información del libro a añadir de una sola vez, en lugar de estar pidiendo dato a dato.

En la consola de Usuario obtenemos lo siguiente:

```
4
Has elegido la opción: 4
Esperando respuesta del servidor
Introduzca los datos según el formato especificado: ISBN-Título-Autor-Precio
9788401022166-Fuego y Sangre-George R.R. Martin-32.90
Libro añadido correctamente

Libro ISBN=9788401022166, titulo=Fuego y Sangre, autor=George R.R. Martin,
precio=32.90
```

Mientras en la consola de la biblioteca nos traza la siguiente información:

```
El Usuario_1 ha solicitado la opción 4
El libro añadido por el Usuario_1 es:
Libro ISBN=9788401022166, titulo=Fuego y Sangre, autor=George R.R. Martin,
precio=32.90
```

- Ahora vamos a iniciar un nuevo Usuario (un nuevo cliente). La consola nos mostrará la conexión de ese nuevo Usuario, en este caso el **Usuario_2.**

```
Petición nº 2 recibida

Estableciendo conexión con: Usuario_2
```

Con este nuevo Usuario, vamos a pedir que nos saque la información sobre el libro que introdujimos en el punto anterior con el **Usuario_1**.

```
2
Has elegido la opción: 2
Esperando respuesta del servidor
Introduzca el Título del libro
Fuego y Sangre
Libro ISBN=9788401022166, titulo=Fuego y Sangre, autor=George R.R. Martin,
precio=32.90
```

En la consola del servidor obtenemos la información de la consulta del **Usuario_2**:

```
El Usuario_2 ha solicitado la opción 2
El Usuario_2 solicita información a través del título: Fuego y Sangre
la info solicitada por el Usuario_2 es:
Libro ISBN=9788401022166, titulo=Fuego y Sangre, autor=George R.R. Martin,
precio=32.90
```

- **Salir de la aplicación:**

Ya solo nos quedaría cerrar las conexiones en ambos clientes

En la consola de ambos clientes seleccionamos la opción 5.

```
5
Has elegido la opción: 5
Esperando respuesta del servidor
Conexión terminada
```

Y la consola del servidor nos informa de que la conexión ha sido cerrada por el lado del cliente y no ha sido un fallo de comunicación, en cuyo caso saldría un error.

```
Conexión cerrada por el Usuario_1

Conexión cerrada por el Usuario_2
```

Si en algún momento, por la razón que sea se interrumpe la conexión por parte del cliente sin ser pulsando la opción de "***Salir de la aplicación***" saldrá el siguiente mensaje de error de ***entrada/salida*** en la consola del servidor capturado por un **try/catch.**

```
Error entrada/salida
java.net.SocketException: Connection reset
	at java.base/sun.nio.ch.NioSocketImpl.implRead(NioSocketImpl.java:323)
	at java.base/sun.nio.ch.NioSocketImpl.read(NioSocketImpl.java:350)
	at java.base/sun.nio.ch.NioSocketImpl$1.read(NioSocketImpl.java:803)
	at java.base/java.net.Socket$SocketInputStream.read(Socket.java:966)
	at java.base/sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:270)
	at java.base/sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:313)
	at java.base/sun.nio.cs.StreamDecoder.read(StreamDecoder.java:188)
	at java.base/java.io.InputStreamReader.read(InputStreamReader.java:177)
	at java.base/java.io.BufferedReader.fill(BufferedReader.java:162)
	at java.base/java.io.BufferedReader.readLine(BufferedReader.java:329)
	at java.base/java.io.BufferedReader.readLine(BufferedReader.java:396)
	at Sockets_e_Hilos/tarea.Hilos.run(Hilos.java:46)
	at java.base/java.lang.Thread.run(Thread.java:833)
```

Si el cliente intenta conectar con el servidor cuando éste se encuentra apagado, o se ha caído, nos dará también un error de ***entrada/salida*** en la consola del cliente correspondiente.

```
        APLICACIÓN USUARIO
---------------------------------
Esperando la respuesta del servidor
Error entrada/salida
java.net.ConnectException: Connection refused: connect
	at java.base/sun.nio.ch.Net.connect0(Native Method)
	at java.base/sun.nio.ch.Net.connect(Net.java:579)
	at java.base/sun.nio.ch.Net.connect(Net.java:568)
	at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:588)
	at java.base/java.net.SocksSocketImpl.connect(SocksSocketImpl.java:327)
	at java.base/java.net.Socket.connect(Socket.java:633)
	at java.base/java.net.Socket.connect(Socket.java:583)
	at Sockets_e_Hilos/tarea.Usuario.main(Usuario.java:32)
```

Cabe decir que solo hemos mostrado dos clientes funcionando a la vez, pero se pueden conectar multitud de ellos, abriendo así todas las conexiones que sean necesarias para los respectivos usuarios.

5 Conclusiones

Este trabajo en conjunto, nos ha parecido muy interesante, puesto que hemos podido aprender y aplicar la funcionalidad de los hilos y los sockets en nuestros programas java, entendiendo cómo funcionan los servidores que frecuentemente visitamos en nuestro día a día, al programar y simular uno de ellos por nosotros mismos.

Además, hemos podido repasar todos los conocimientos adquiridos durante el curso anterior en lo referente al lenguaje de Java, lo que nos ha permitido afianzarlos y reforzar aquellos que ya conocíamos.

Asimismo, nos hemos enfrentado a varios retos mientras avanzando en nuestro proyecto, lo que también nos aporta una experiencia de cara a posibles problemas en un futuro al realizar este tipo de acciones. Ha sido interesante que cada integrante del grupo, aportara sus propias pruebas y argumentos de todos y cada uno de los problemas que se iban presentado durante la realización de la activad, con el fin de comparar y poder debatir acerca de cuál sería la forma más adecuada de realizar ciertas acciones.

Por otro lado, volvemos a señalar, dos partes que han sido de suma importancia durante este proyecto. La primera de ellas y donde más hemos aprendido los unos de los otros, ha sido al realizar la funcionalidad principal del programa de manera conjunta (programación de Sockets e Hilos), mientras que la segunda parte, ha sido la de decidir, cuál sería el formato de la información entre Usuario y Servidor, para poder establecer una buena comunicación entre ambos durante el funcionamiento de la aplicación.

Algunas de las decisiones más importantes que hemos tomado en conjunto, han sido acerca de la presentación del proyecto, utilizando en todo momento buenas prácticas y que todo quedará de la forma más clara y ordenada. También hemos debatido acerca de cómo implementar ciertos métodos, así como la funcionalidad esperada de los mismos. Aunque algunas decisiones tuvieran más peso que otras, al final ha sido fundamental ponernos de acuerdo, escuchando a todas las partes para sacar adelante un buen proyecto.

Al final esta metodología nos ayuda a crecer, en lo referente al trabajo en equipo, recordándonos lo importante que resulta la puesta en común de todas las ideas, puesto que, esta estrategia de trabajo, enriquece a todos los integrantes del equipo y genera que el proyecto a realizar sea mucho más dinámico e interactivo.