

HITO GRUPAL

PG SyP T1



Isla P y Sergio C

16/11/2022



ÍNDICE

TECNOLOGÍAS UTILIZADAS	2
CAPTURAS DE PANTALLA	3
EXPLICACIÓN DE LAS PARTES MÁS RELEVANTES	6



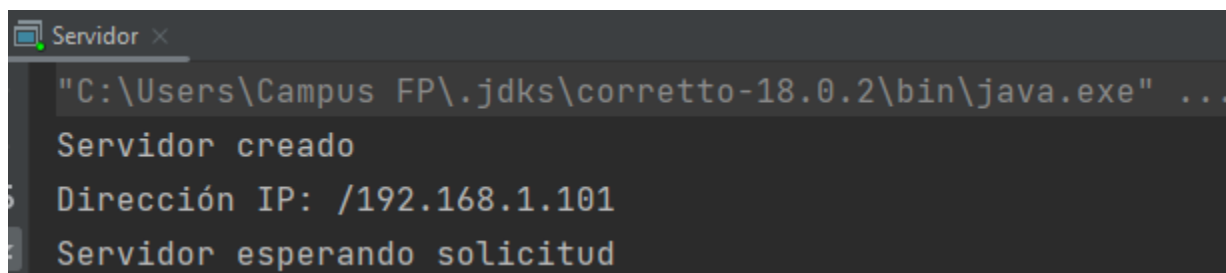
TECNOLOGÍAS UTILIZADAS

- Lenguaje utilizado : Java.
- IDE : IntelliJ Ultimate.
- Tecnologías : Archivos csv, leer archivos csv en java, Map de Java donde almacenamos el contenido del archivo csv, conexiones a bases de datos SQL, Sockets de java, Streams de java, Thread de java, entre otras.
- Versión de Java : Amazon Corretto 18.0.2.
- Librerías : Junit, OpenCSV, Maven-archetype-quickstart, entre otras.
- Encoding : UTF-8.
- Equipo : Windows 10.

CAPTURAS DE PANTALLA

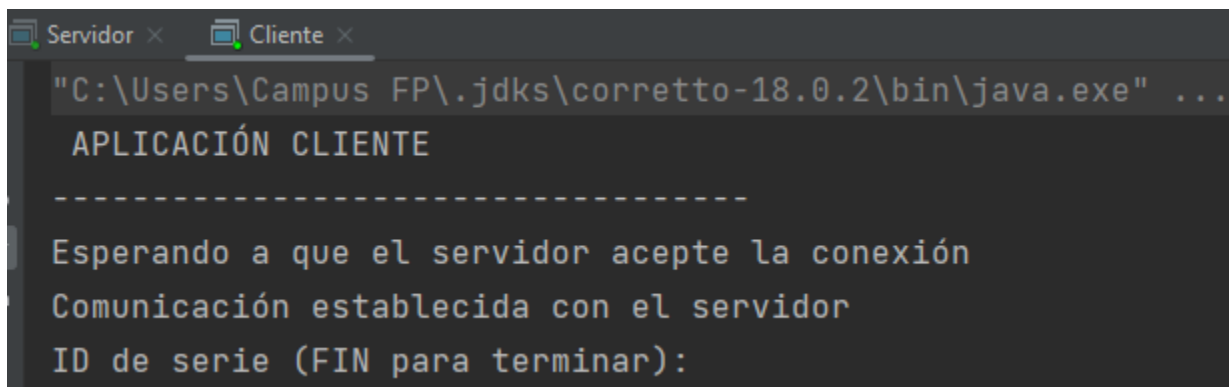
El programa funciona de manera que iniciamos la clase 'Servidor', iniciamos la clase 'Cliente' y este debe establecer conexión con el servidor (en la consola de la clase 'Servidor' muestra si la clase 'Cliente' se ha conectado correctamente). Una vez se han conectado entre ellos el cliente por consola le escribe un número del 1 al 10 para consultar las series disponibles, el servidor le envía la respuesta de la petición hecha y el cliente puede seguir preguntando al servidor por series o escribir "FIN" y cerrar la conexión. Cuando se cierra la conexión de 'Cliente' el servidor sigue en escucha de más peticiones (multitarea).

- Clase servidor :



```
"C:\Users\Campus FP\.jdk\corretto-18.0.2\bin\java.exe" ...
Servidor creado
Dirección IP: /192.168.1.101
Servidor esperando solicitud
```

- Clase Cliente establece conexión con el servidor:



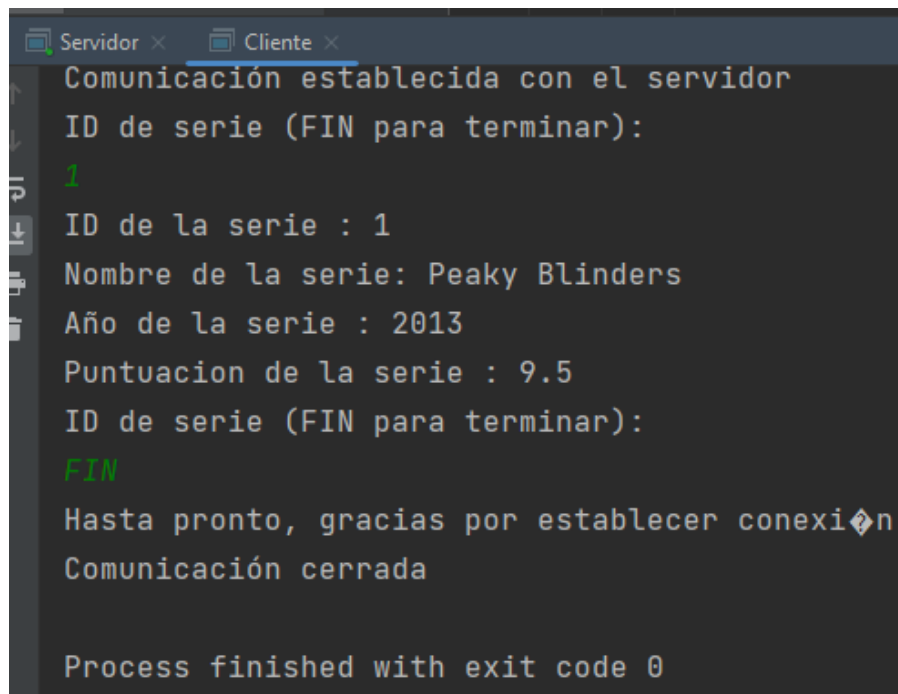
```
"C:\Users\Campus FP\.jdk\corretto-18.0.2\bin\java.exe" ...
APLICACIÓN CLIENTE
-----
Esperando a que el servidor acepte la conexión
Comunicación establecida con el servidor
ID de serie (FIN para terminar):
```

- Clase Servidor acepta la conexión del cliente :

```
Servidor x Cliente x
"C:\Users\Campus FP\.jdk\corretto-18.0.2\bin\java.exe" ...
Servidor creado
Dirección IP: /192.168.1.101
Servidor esperando solicitud
Cliente conectado
Servidor esperando solicitud
Estableciendo comunicación con Cliente1
|
```

Clase Cliente solicita serie al servidor, respuesta del servidor y cerrar conexión :

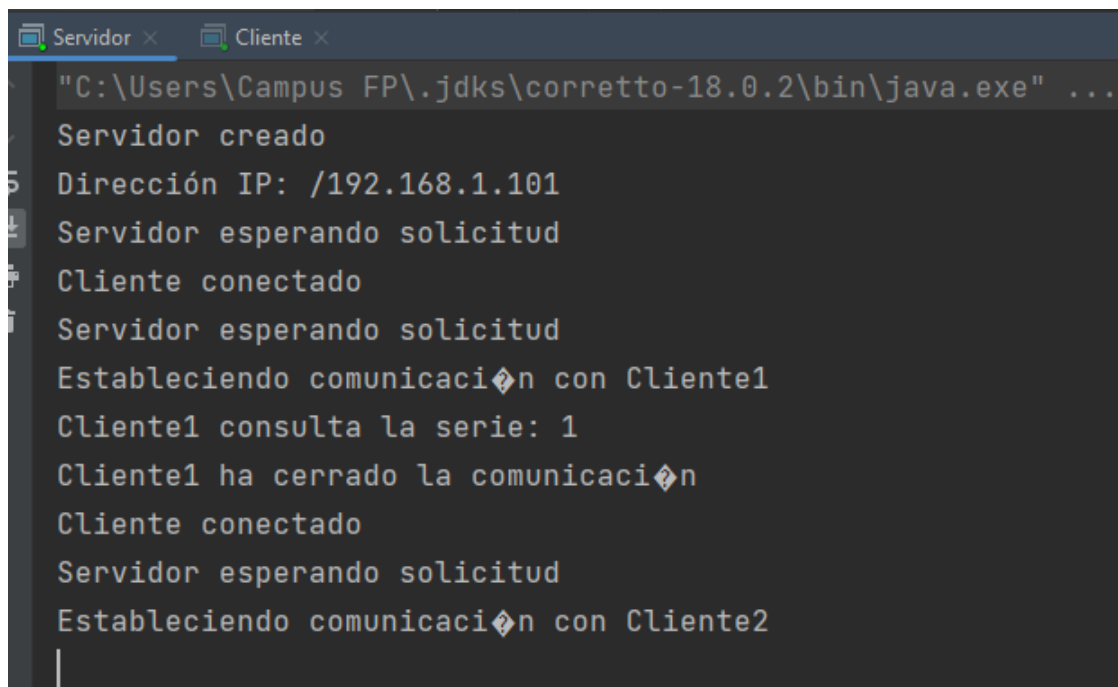
```
Servidor x Cliente x
"C:\Users\Campus FP\.jdk\corretto-18.0.2\bin\java.exe" ...
APLICACIÓN CLIENTE
-----
Esperando a que el servidor acepte la conexión
Comunicación establecida con el servidor
ID de serie (FIN para terminar):
1
ID de la serie : 1
Nombre de la serie: Peaky Blinders
Año de la serie : 2013
Puntuacion de la serie : 9.5
ID de serie (FIN para terminar):
|
```



```
Servidor x Cliente x
Comunicación establecida con el servidor
ID de serie (FIN para terminar):
1
ID de la serie : 1
Nombre de la serie: Peakys Blinders
Año de la serie : 2013
Puntuacion de la serie : 9.5
ID de serie (FIN para terminar):
FIN
Hasta pronto, gracias por establecer conexión
Comunicación cerrada

Process finished with exit code 0
```

Clase servidor sigue en escucha de más peticiones (para demostrarlo he iniciado otra clase 'Cliente' la cual será 'Cliente2'):



```
Servidor x Cliente x
"C:\Users\Campus FP\.jdk\corretto-18.0.2\bin\java.exe" ...
Servidor creado
Dirección IP: /192.168.1.101
Servidor esperando solicitud
Cliente conectado
Servidor esperando solicitud
Estableciendo comunicación con Cliente1
Cliente1 consulta la serie: 1
Cliente1 ha cerrado la comunicación
Cliente conectado
Servidor esperando solicitud
Estableciendo comunicación con Cliente2
|
```

EXPLICACIÓN DE LAS PARTES MÁS RELEVANTES

Como partes más relevantes del código señalaría la lectura del archivo csv, la conexión entre Cliente/Servidor (Sockets) y la multitarea que permite que el servidor quede en escucha de más peticiones (Thread).

- Archivo CSV :

Primero hemos creado un archivo CSV donde hemos almacenado las series (id, nombre, año y puntuación) y los campos están delimitados por ‘,’.

```
1,Peaky Blinders,2013,9.5
2,The Walking Dead,2010, 8.7
3,Las chicas Gilmore,2000,7
4,La que se avecina,2007,7.8
5,Agente Carter,2015,8.2
6,Jessica Jones,2015,3.6
7,La bruja escarlata,2021,7
8,Vikings,2013,7
9,Stranger Things,2016,8.2
10,F1 Drive to Survive,2019,7.5|
```

A continuación, añadimos la librería ‘opencsv’ a nuestro proyecto para leer el archivo csv.

```
<!-- https://mvnrepository.com/artifact/com.opencsv/opencsv -->
<dependency>
  <groupId>com.opencsv</groupId>
  <artifactId>opencsv</artifactId>
  <version>5.7.1</version>
</dependency>
```


Ahora, desarrollamos el código. El código está hecho en la clase servidor el cual lee el archivo CSV y añade el contenido a un 'Map' debido a que el usuario consultará las series mediante el 'ID' de la serie.

Creamos la variable donde guardamos el fichero y el 'Map' para guardar su contenido. Mediante un 'Scanner' leemos línea a línea el archivo CSV y por cada línea que contenga llamamos al método 'fragmentarLinea'.

```
File fichero = new File( pathname: "series.csv");
seriesMap = new HashMap<Integer, Serie>();
Scanner sc;
try {
    sc = new Scanner(fichero);
    while (sc.hasNextLine()) {
        String linea = sc.nextLine();
        fragmentarLinea(linea);
    }
    sc.close();
} catch (FileNotFoundException e) {
    System.out.println("No se ha podido leer el archivo");
    System.out.println(e.getMessage());
}
```

Metodo FragmentarLinea :

```
public static void fragmentarLinea(String linea) {
    Scanner sc = new Scanner(linea);
    sc.useDelimiter( pattern: ",");
    Integer id = Integer.parseInt(sc.next());
    String nombre_series = sc.next();
    int year = Integer.parseInt(sc.next());
    float puntuacion = Float.parseFloat(sc.next());
    Serie s = new Serie(id,nombre_series,year,puntuacion);
    seriesMap.put(id,s);
}
```

El método ‘fragmentarLinea’ lo que hace es que lee la línea del archivo csv, establece la ‘,’ como delimitador de los campos, establece los diferentes campos que va a contener, crea un objeto serie de la clase ‘Serie’ con los campos como atributos y lo añade al ‘Map’ de series (esto por cada línea que contenga el archivo csv).

Ya tenemos un ‘Map’ de series al que el cliente acudirá para consultar series.

- Conexión Cliente/Servidor :

La conexión Cliente/Servidor la creamos a partir de 'Sockets'. Este socket está del lado del servidor. Creamos el socket, establecemos la dirección IP y la añadimos al servidor.

```
try {
    //CREAMOS LA CONEXION POR RED ENTRE CLIENTE/SERVIDOR
    //Socket del extremo del servidor
    ServerSocket servidor = new ServerSocket();
    //Definimos la direccion IP y puerto del servidor.
    InetSocketAddress direccion = new InetSocketAddress( hostname: "192.168.1.101", port: 2000);
    //Asignamos al socket del servidor la direccion IP y puerto mediante el metodo '.bind()'
    servidor.bind(direccion);
    System.out.println("Servidor creado");
    //Comprobamos la direccion.
    System.out.println("Dirección IP: " + direccion.getAddress());
    Scanner sc2 = new Scanner(System.in);
}
```

Aquí, aceptamos la conexión del cliente. Creamos el socket para el cliente y le especificamos que nuestro socket 'servidor' acepta la conexión del cliente. Creamos un nuevo 'hiloEscuchador' el cual tendrá como parámetros a destacar el hilo para permitir multitarea, el 'Map' de series y los streams para intercambiar datos.

```
while (true) {
    System.out.println("Servidor esperando solicitud");
    Socket conexionCliente = servidor.accept();
    System.out.println("Cliente conectado");
    new HiloEscuchador(conexionCliente, seriesMap);
}
```

Para el socket 'Cliente' hacemos lo siguiente. Creamos el socket, especificamos la ruta y puerto del servidor donde nos conectaremos y con el método '.connect()' establecemos conexión con el servidor.

```
Socket cliente = new Socket();
InetSocketAddress direccionServidor = new InetSocketAddress( hostname: "192.168.1.101", port: 2000);
System.out.println("Esperando a que el servidor acepte la conexión");
cliente.connect(direccionServidor);
// Conectamos con el servidor.
System.out.println("Comunicación establecida con el servidor");
```

En la clase 'HiloEscuchador' es donde evaluaremos los datos que introduce el cliente para solicitar las series mediante bucles y estructuras de control.

```
salida = new ObjectOutputStream(conexionCliente.getOutputStream());
entrada = new ObjectInputStream(conexionCliente.getInputStream());
String id;
do {
    id = (String) entrada.readObject();
    if (id.trim().equals("FIN")) {
        salida.writeObject("Hasta pronto, gracias por establecer conexión");
        System.out.println(hilo.getName() + " ha cerrado la comunicación");
    } else {
        System.out.println(hilo.getName() + " consulta la serie: " + id);
        // Enviamos el objeto correspondiente al alumno consultado.
        try{
            Integer idInt = Integer.parseInt(id);
            Serie mensaje = seriesMap.get(idInt);
            if (mensaje==null) {
                salida.writeObject("Serie no encontrado");
            }
            else {
                salida.writeObject(mensaje);
            }
        }catch(IOException e){
            e.printStackTrace();
            System.out.println("fin - Debe ser en mayusculas");
        }
    }
} while ((!id.trim().equals("FIN")));
entrada.close();
salida.close();
conexionCliente.close();
```

- Multitarea :

La multitarea la creamos mediante hilos. Cada cliente que se conecte tendrá un hilo diferente por lo que podemos conectar más de un cliente a la vez y permitir que nuestro servidor una vez cierre la conexión de un cliente siga en escucha. Para ello en la clase 'HiloEscuchador' establecemos los hilos.

```
//CONSTRUCTOR
1 usage
public HiloEscuchador(Socket cliente, Map<Integer, Serie> seriesMap) {
    numCliente++;
    hilo = new Thread( target: this, name: "Cliente" + numCliente);
    this.conexionCliente = cliente;
    this.seriesMap = seriesMap;
    hilo.start();
}
```

En esta línea es donde le pasamos el socket cliente y el 'Map' de series para que la clase 'HiloEscuchador' se encargue de la multitarea y de evaluar la entrada de datos por consola del cliente.

```
new HiloEscuchador(conexionCliente, seriesMap);
```

ESPECIFICACIONES ACTIVIDAD

Este HITO consiste en desarrollar una aplicación Java cliente/servidor donde el servidor pueda atender múltiples peticiones de clientes simultáneos.

La aplicación se desarrollará usando Sockets, hilos y streams (flujos de datos).

El servidor, para atender las peticiones tendrá que consultar un fichero de texto en formato csv con estructura libre. El servidor responderá enviando un objeto.

El cliente se comunica con el servidor para solicitar datos que se encuentran en el fichero de texto.

▪ La aplicación contará con tres capas:

1. **Capa de acceso a datos:** donde se realiza la lectura al archivo de texto para ayudar al servidor a atender las solicitudes de los clientes.
2. **Capa servidor:** que atenderá solicitudes de los clientes que desean obtener información. Ejemplo: el servidor podría atender la solicitud de un cliente que quiere información sobre libros de java, el servidor recibe dicha solicitud, hace uso de la capa de datos y envía al cliente la información solicitada.
3. **Capa cliente:** que se comunica con el servidor enviando una clave de búsqueda. Ejemplo: el cliente envía la palabra Java con el fin de obtener información sobre los libros de Java disponibles en la librería.