

AE-1. Hilos y Sockets

Sergio Palmero — Christian Casado

2º DAM

Objetivos.

Aprender a trabajar de manera practica con los hilos y los sockets en Java.

Para este trabajo hemos utilizado Git y GitHub para control de versiones, poder trabajar cada uno de manera independiente y así después sincronizar nuestro código corrigiendo posibles errores y bugs.

Aquí dejamos el enlace a Github donde está todo el código, todos los commits hechos y todo el avance del proyecto:

https://github.com/sergiopalmero87/Act_Hilos_Sockets

https://github.com/casadochristian/Hilos_y_sockets

Algunos problemas que tuvimos fue que Sergio hizo el proyecto en IntelliJ Idea y Christian cuando hizo un git clone del proyecto de Sergio, no fue capaz de abrirlo en Eclipse, así que por eso hay dos enlaces a Github y no uno con diferentes ramas.

Vamos a mostrar mediante diapositivas el código fuente:

Servidor:

The screenshot shows the Java code for the `ServidorPelículas` class within an IDE. The code includes fields for `id`, `título`, `nombreDirector`, and `precio`, a constructor, and an `equals` method. The code is annotated with comments and developer names like `sergiopalmero`.

```
public class ServidorPelículas {
    int id;
    String título, nombreDirector;
    double precio;

    public static final int PUERTO = 12345;

    static List<ServidorPelículas> listaPelículas = new ArrayList<>();

    public ServidorPelículas(int id, String título, String nombreDirector, double precio) {
        this.id = id;
        this.título = título;
        this.nombreDirector = nombreDirector;
        this.precio = precio;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        ServidorPelículas that = (ServidorPelículas) o;
        return id == that.id && Double.compare(precio, that.precio) == 0 && Objects.equals(título, that.título) && Objects.equals(nombreDirector, that.nombreDirector);
    }
}
```

The screenshot shows the `main` method and the server setup logic within the `ServidorPelículas` class. It creates a list of movies, initializes a server socket, and binds it to port 12345.

```
@Override
public int hashCode() {
    return Objects.hash(id, título, nombreDirector, precio);
}

@Override
public String toString() {
    return "ServidorPelículas{" +
        "id=" + id +
        ", título='" + título + '\'' +
        ", nombreDirector='" + nombreDirector + '\'' +
        ", precio=" + precio +
        '}';
}

public static void main(String[] args) {
    System.out.println("-----SERVIDOR-----");

    listaPelículas.add(new ServidorPelículas(1, "Pelicula 1", "Director 1", 12.0));
    listaPelículas.add(new ServidorPelículas(2, "Pelicula 2", "Director 2", 14.0));
    listaPelículas.add(new ServidorPelículas(3, "Pelicula 3", "Director 3", 16.0));
    listaPelículas.add(new ServidorPelículas(4, "Pelicula 4", "Director 4", 18.0));
    listaPelículas.add(new ServidorPelículas(5, "Pelicula 5", "Director 5", 20.0));

    try(ServerSocket servidor = new ServerSocket()){
        InetSocketAddress dirección = new InetSocketAddress(PUERTO);
        servidor.bind(dirección);
        System.out.println("Servidor escuchando por puerto " + PUERTO);

        //Hacemos que el servidor este siempre escuchando hasta que el cliente nos mande 5.
        //Cada vez que se acepte una conexión se crea un nuevo hilo y toda la lógica se hace en ese hilo
        //así dejamos al servidor libre para que pueda seguir recibiendo peticiones.
    }
}
```

The screenshot shows a Java code editor interface with a dark theme. The main area displays a class named `ServidorPeliculas` in a file named `ServidorPeliculas.java`. The code implements a server socket that listens for incoming connections and creates a new thread for each client. It also handles exceptions and prints logs to the console.

```
public class ServidorPeliculas {
    public static void main(String[] args) {
        listaPeliculas.add(new Pelicula(1, "Pelicula 1", "Director 1", 10.0));
        listaPeliculas.add(new Pelicula(2, "Pelicula 2", "Director 2", 12.0));
        listaPeliculas.add(new Pelicula(3, "Pelicula 3", "Director 3", 15.0));
        listaPeliculas.add(new Pelicula(4, "Pelicula 4", "Director 4", 18.0));
        listaPeliculas.add(new Pelicula(5, "Pelicula 5", "Director 5", 20.0));

        try(ServerSocket servidor = new ServerSocket()){
            InetSocketAddress direccion = new InetSocketAddress(PUERTO);
            servidor.bind(direccion);
            System.out.println("Servidor escuchando por puerto " + PUERTO);

            //Hacemos que el servidor este siempre escuchando hasta que el cliente nos mande 5.
            //Cada vez que se acepte una conexión se crea un nuevo hilo y toda la lógica se hace en ese hilo
            //así dejamos al servidor libre para que pueda seguir recibiendo peticiones.
            while (true){
                Socket socketAlCliente = servidor.accept();
                System.out.println("El servidor acepta la petición del cliente");
                new Hilo(socketAlCliente);
            }
        }catch (IOException e){
            System.err.println("SERVIDOR: Error de entrada/salida");
            e.printStackTrace();
        }catch(Exception e){
            System.err.println("SERVIDOR: Error");
            e.printStackTrace();
        }
    }
}
```

Cliente:

The screenshot shows the Java code for the Client application. The code is written in a dark-themed IDE. It includes imports for java.util.Scanner and java.io.*. The class Cliente has a static final port number (PUERTO) and a static final IP address (IP_SERVER). The main method establishes a connection to the server at IP_SERVER:PUERTO, prints a message to the client, and then enters a loop where it prints a menu and reads user input. It handles five options: 1. Consulta por ID, 2. Consulta por Titulo, 3. Consulta por Nombre del director, 4. Añadir pelicula, and 5. Salir.

```
11  public class Cliente {
12
13     2 usages
14     public static final int PUERTO = 12345;
15
16     2 usages
17     public static final String IP_SERVER = "LocalHost";
18
19     ▲ sergiopalmero
20     public static void main(String[] args) throws IOException {
21
22         InetSocketAddress direccion = new InetSocketAddress(IP_SERVER, PUERTO);
23         System.out.println("-----CLIENTE-----");
24         // Try-catch with recursos para que el scanner se cierre solo.
25         try(Scanner sc = new Scanner(System.in)){
26
27             System.out.println("CLIENTE: Esperando a que el servidor acepte la conexión");
28             Socket socketAlServidor = new Socket();
29             socketAlServidor.connect(direccion);
30             System.out.println("CLIENTE: Conexión establecida... a " + IP_SERVER + " por el puerto " + PUERTO);
31
32             //Le mandamos los datos al servidor
33             PrintStream salida = new PrintStream(socketAlServidor.getOutputStream());
34
35             //Recibimos los datos del servidor
36             InputStreamReader entrada = new InputStreamReader(socketAlServidor.getInputStream());
37
38             //Procesamos los datos que nos manda el servidor
39             BufferedReader entradaBuffer = new BufferedReader(entrada);
40
41             boolean continuar = true;
42             do {
43                 String opcion;
44                 //Mostramos el menu al cliente
45                 System.out.println("-----");
46                 System.out.println("Elige una opción: ");
47                 System.out.println("1. Película por ID: ");
48                 System.out.println("2. Película por Título: ");
49                 System.out.println("3. Película por Nombre del director: ");
50                 System.out.println("4. Añadir película: ");
51                 System.out.println("5. Salir: ");
52                 System.out.println("-----");
53
54                 opcion = sc.nextLine();
55                 String consulta;
56
57                 switch (opcion){
58                     case "1":
59                         System.out.println("Consulta por ID:");
60                         salida.print(opcion);
61                         System.out.println("Escribe el id: ");
62                         consulta = sc.nextLine();
63                         salida.println(consulta);
64                         break;
65                     case "2":
66                         System.out.println("Consulta por título: ");
67                         salida.print(opcion);
68                         System.out.println("Escribe el título: ");
69                         consulta = sc.nextLine();
70                         salida.println(consulta);
71                         break;
72                     case "3":
73                         System.out.println("Consulta por nombre del director: ");
74                         salida.print(opcion);
75                         System.out.println("Escribe el nombre del director: ");
76                         consulta = sc.nextLine();
77                         salida.println(consulta);
78                         break;
79                 }
80             } while(continuar);
81         }
82     }
83 }
```

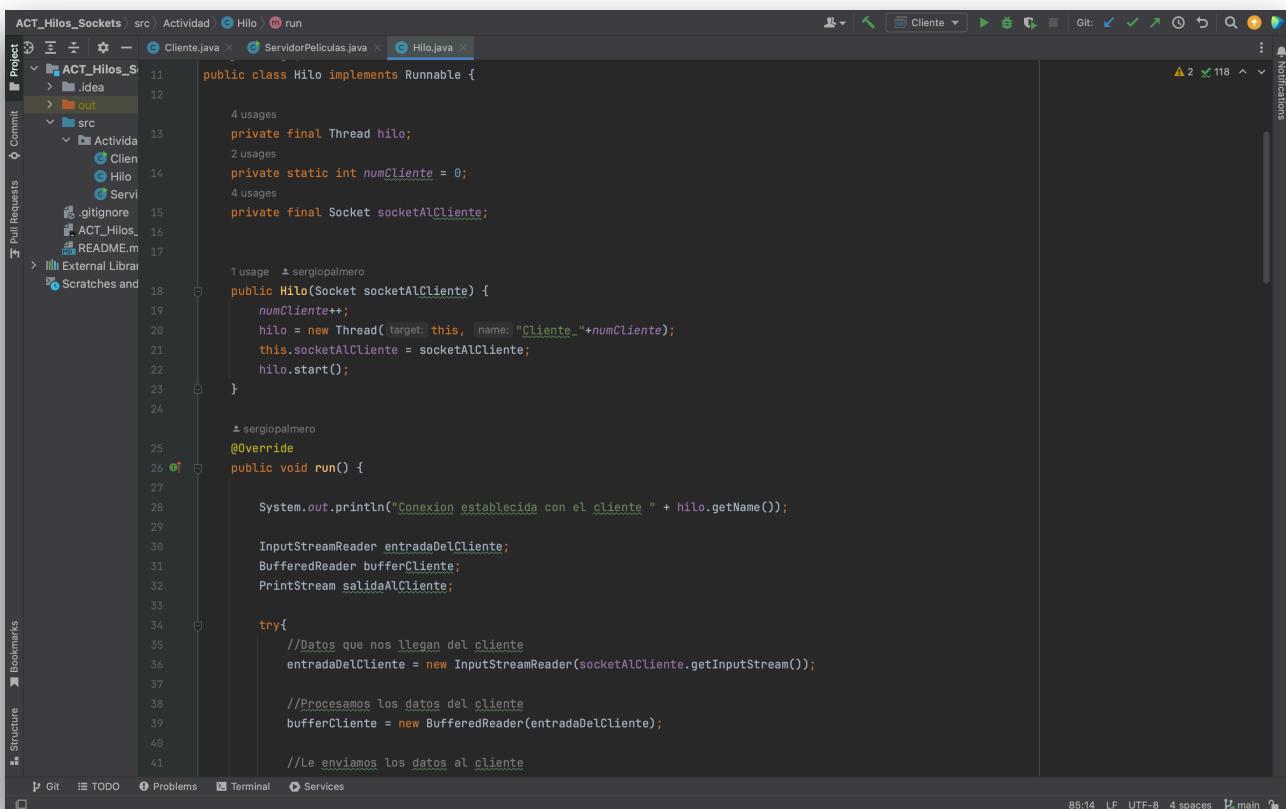
This screenshot shows the same Client.java code as the previous one, but with additional logic added. The code now includes a loop that continues until the user chooses option 5 (Salir). Inside this loop, it prints a menu and reads user input. It handles five options: 1. Consulta por ID, 2. Consulta por Titulo, 3. Consulta por Nombre del director, 4. Añadir pelicula, and 5. Salir.

```
38     boolean continuar = true;
39     do {
40         String opcion;
41         //Mostramos el menu al cliente
42         System.out.println("-----");
43         System.out.println("Elige una opción: ");
44         System.out.println("1. Película por ID: ");
45         System.out.println("2. Película por Título: ");
46         System.out.println("3. Película por Nombre del director: ");
47         System.out.println("4. Añadir película: ");
48         System.out.println("5. Salir: ");
49         System.out.println("-----");
50
51         opcion = sc.nextLine();
52         String consulta;
53
54
55         switch (opcion){
56             case "1":
57                 System.out.println("Consulta por ID:");
58                 salida.print(opcion);
59                 System.out.println("Escribe el id: ");
60                 consulta = sc.nextLine();
61                 salida.println(consulta);
62                 break;
63             case "2":
64                 System.out.println("Consulta por título: ");
65                 salida.print(opcion);
66                 System.out.println("Escribe el título: ");
67                 consulta = sc.nextLine();
68                 salida.println(consulta);
69                 break;
70             case "3":
71                 System.out.println("Consulta por nombre del director: ");
72                 salida.print(opcion);
73                 System.out.println("Escribe el nombre del director: ");
74                 consulta = sc.nextLine();
75                 salida.println(consulta);
76                 break;
77         }
78     } while(continuar);
79 }
```

```
case "4":  
    System.out.println("AÑADIENDO NUEVA PELICULA---");  
    salida.println(opcion);  
    System.out.println("Escribe los datos para añadir una nueva pelicula");  
    salida.println(opcion);  
    System.out.println("ID: ");  
    String id = sc.nextLine();  
    //salida.println(id);  
  
    System.out.println("Titulo: ");  
    String titulo = sc.nextLine();  
    //salida.println(titulo);  
    //Para saltar de linea y acceder al siguiente scanner.  
  
    System.out.println("Nombre del director: ");  
    String nombreDirector = sc.nextLine();  
    //salida.println(nombreDirector);  
  
    System.out.println("Precio: ");  
    String precio = sc.nextLine();  
    //salida.println(precio);  
    //sc.nextLine();  
  
    String info = id + "-" + titulo + "-" + nombreDirector + "-" + precio;  
    salida.println(info);  
    break;  
  
case "5":  
    salida.println(opcion);  
    continuar = false;  
    break;  
default:  
    System.out.println("Opción no válida. Por favor, introduce una opción entre 1 y 5.");  
    break;
```

```
        break;  
  
    }  
  
    //Le mandamos el texto al servidor para que lo procese con salida.println(texto)  
  
    System.out.println("Consulta enviada al servidor");  
    System.out.println("Cliente esperando respuesta del servidor...");  
  
    //Procesamos los datos del servidor  
    String respuestaDelServidor = entradaBuffer.readLine();  
  
    System.out.println("CLIENTE: Servidor responde: " + respuestaDelServidor);  
  
} while (continuar);  
//Cerramos conexión con el servidor  
socketAlServidor.close();  
  
} catch (IOException e) {  
    throw new RuntimeException(e);  
} catch (Exception e) {  
    System.err.println("CLIENTE: Error -> " + e);  
    e.printStackTrace();  
}  
  
System.out.println("Cliente: Fin del programa");  
}
```

Hilo:



```
public class Hilo implements Runnable {

    private final Thread hilo;
    private static int numCliente = 0;
    private final Socket socketAlCliente;

    public Hilo(Socket socketAlCliente) {
        numCliente++;
        hilo = new Thread( target: this, name: "Cliente_"+numCliente );
        this.socketAlCliente = socketAlCliente;
        hilo.start();
    }

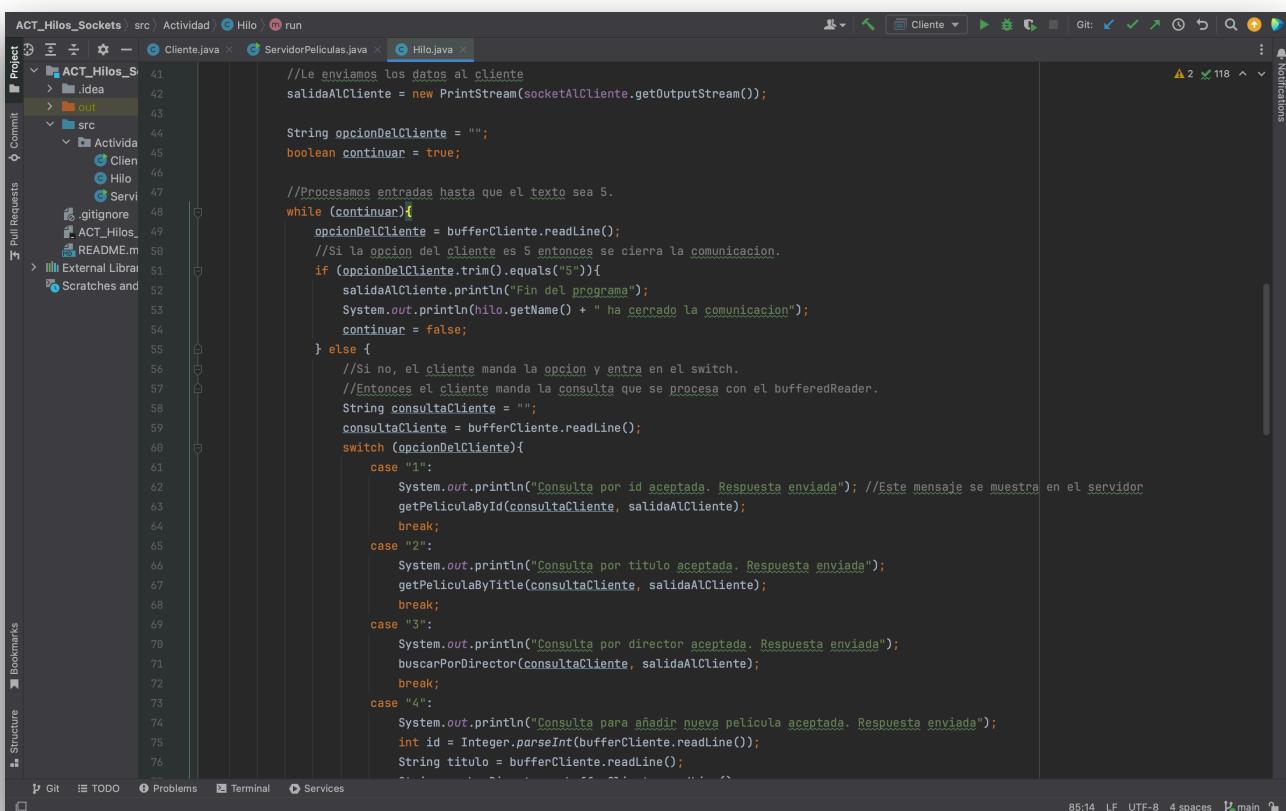
    @Override
    public void run() {
        System.out.println("Conexion establecida con el cliente " + hilo.getName());

        InputStreamReader entradaDelCliente;
        BufferedReader bufferCliente;
        PrintStream salidaAlCliente;

        try{
            //Datos que nos llegan del cliente
            entradaDelCliente = new InputStreamReader(socketAlCliente.getInputStream());

            //Procesamos los datos del cliente
            bufferCliente = new BufferedReader(entradaDelCliente);

            //Le enviamos los datos al cliente
        }
    }
}
```



```
//Le enviamos los datos al cliente
salidaAlCliente = new PrintStream(socketAlCliente.getOutputStream());

String opcionDelCliente = "";
boolean continuar = true;

//Procesamos entradas hasta que el texto sea 5.
while (continuar){
    opcionDelCliente = bufferCliente.readLine();
    //Si la opcion del cliente es 5 entonces se cierra la comunicacion.
    if (opcionDelCliente.trim().equals("5")){
        salidaAlCliente.println("Fin del programa");
        System.out.println(hilo.getName() + " ha cerrado la comunicacion");
        continuar = false;
    } else {
        //Si no, el cliente manda la opcion y entra en el switch.
        //Entonces el cliente manda la consulta que se procesa con el bufferedReader.
        String consultaCliente = "";
        consultaCliente = bufferCliente.readLine();
        switch (opcionDelCliente){
            case "1":
                System.out.println("Consulta por id aceptada. Respuesta enviada"); //Este mensaje se muestra en el servidor
                getPeliculaById(consultaCliente, salidaAlCliente);
                break;
            case "2":
                System.out.println("Consulta por titulo aceptada. Respuesta enviada");
                getPeliculaByTitle(consultaCliente, salidaAlCliente);
                break;
            case "3":
                System.out.println("Consulta por director aceptada. Respuesta enviada");
                buscarPorDirector(consultaCliente, salidaAlCliente);
                break;
            case "4":
                System.out.println("Consulta para añadir nueva pelicula aceptada. Respuesta enviada");
                int id = Integer.parseInt(bufferCliente.readLine());
                String titulo = bufferCliente.readLine();
                addNewMovie(id, titulo, salidaAlCliente);
        }
    }
}
```

```
default:
    salidaAlCliente.println("Error. Número de consulta incorrecto.");
}

//Cerramos conexión con el cliente
socketALCliente.close();

} catch (IOException e) {
    throw new RuntimeException(e);
}

//Logica para responder al cliente

//Funciones para buscar películas dentro de la lista.
//Reciben dos parámetros: El dato a consultar y la salida hacia el cliente.
1 usage: ▲ sergiopalmero
private synchronized void getPelículaById(String idConsulta, PrintStream salidaAlCliente) {
    for (ServidorPelículas p : listaPelículas) {
        if (String.valueOf(p.id).equals(idConsulta)) {
            salidaAlCliente.println(p);
            return; // Salir del bucle una vez que se encuentra una coincidencia.
        }
    }
    salidaAlCliente.println("No se encontró ninguna película con el ID especificado");
}

1 usage: ▲ sergiopalmero
private synchronized void getPelículaByTitle(String títuloConsulta, PrintStream salidaAlCliente) {
    List<ServidorPelículas> películas = new ArrayList<>();
    for (ServidorPelículas p : listaPelículas) {
        if (p.título.equalsIgnoreCase(títuloConsulta)) {
            películas.add(p);
        }
    }
    if (películas.isEmpty()) {

```

```
        List<ServidorPelículas> películas = new ArrayList<>();
        for (ServidorPelículas p : listaPelículas) {
            if (p.título.equalsIgnoreCase(títuloConsulta)) {
                películas.add(p);
            }
        }
        if (películas.isEmpty()) {
            salidaAlCliente.println("No se encontró ninguna película con el título especificado");
        } else {
            salidaAlCliente.println(películas);
        }
    }

1 usage: ▲ sergiopalmero
private synchronized void buscarPorDirector(String directorConsulta, PrintStream salidaAlCliente) {
    List<ServidorPelículas> películasDirector = new ArrayList<>();
    for (ServidorPelículas p : listaPelículas) {
        if (directorConsulta.equalsIgnoreCase(p.nombreDirector)) {
            películasDirector.add(p);
        }
    }
    if (películasDirector.isEmpty()) {
        salidaAlCliente.println("No se encontró ninguna película con el nombre del director especificado");
    } else {
        salidaAlCliente.println(películasDirector);
    }
}

1 usage: ▲ sergiopalmero
private synchronized void addPelícula(int id, String título, String nombreDirector, double precio, PrintStream salidaAlCliente) {
    ServidorPelículas nuevaPelícula = new ServidorPelículas(id, título, nombreDirector, precio);
    listaPelículas.add(nuevaPelícula);
    salidaAlCliente.println("La película se ha añadido con éxito:");
    salidaAlCliente.println(listaPelículas.toString());
}
```

Pasamos a mostrar también las diapositivas para demostrar que todo funciona.

Aquí mostramos cómo el servidor se levanta y el cliente se conecta a él. También se muestra el menú al cliente para que este pueda elegir la opción que desea.

The screenshot shows the IntelliJ IDEA interface with the project 'ACT_Hilos_Sockets' open. The 'Run' tool window displays the command line for running the server:

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=49522:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/sergionalmoropez/Desktop/PSP/ACT_Hilos_Sockets/out/production /ACT_Hilos_Sockets Actividad.ServidorPeliculas
```

Below this, the output shows the server listening on port 12345 and accepting a client connection:

```
-----SERVIDOR-----
Servidor escuchando por puerto 12345
El servidor acepta la petición del cliente
Conexion establecida con el cliente Cliente_1
```

The 'Client' terminal window shows the client connecting to the server on port 12345:

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=49525:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/sergionalmoropez/Desktop/PSP/ACT_Hilos_Sockets/out/production /ACT_Hilos_Sockets Actividad.Cliente
-----CLIENTE-----
CLIENTE: Esperando a que el servidor acepte la conexión
CLIENTE: Conexion establecida... a LocalHost por el puerto 12345
-----
Elige una opción:
1. Pelicula por ID:
2. Pelicula por Título:
3. Pelicula por Nombre del director:
4. Añadir pelicula:
5. Salir:
```

Cuando el cliente elige una opción del menú, esta se manda al servidor. En el hilo que se crea cuando el servidor acepta la petición, se realiza la lógica necesaria para devolver la respuesta al cliente.

Aquí vemos la consulta por ID.

```
ACT_Hilos_Sockets | src | Actividad | ServidorPelículas | precio
Project Run: Cliente.java | ServidorPelículas.java | Hilo.java
Run: ServidorPelículas
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=49525:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/sergiopalmerolopez/Desktop/PSP/ACT_Hilos_Sockets/out/production/ACT_Hilos_Sockets Actividad.ServidorPelículas
-----SERVIDOR-----
Servidor escuchando por puerto 12345
El servidor acepta la petición del cliente
Conexión establecida con el cliente Cliente_1
Consulta por id aceptada. Respuesta enviada

-----CLIENTE-----
CLIENTE: Esperando a que el servidor acepte la conexión
CLIENTE: Conexión establecida... a LocalHost por el puerto 12345
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
1
Consulta por ID:
Escribe el id:
1
Consulta enviada al servidor
Cliente esperando respuesta del servidor...
CLIENTE: Servidor responde: ServidorPelículas{id=1, título='Película 1', nombreDirector='Director 1', precio=12.0}
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
```

Aquí vemos la consulta por título.

The screenshot shows the IntelliJ IDEA interface with a Java project named "ACT_Hilos_Sockets". The "Project" tool window on the left lists three files: "ServidorPeliculas.java", "Cliente.java", and "Hilo.java". The "Run" tool window on the right displays the execution of the "ServidorPeliculas" class. The output shows:

```
Run: ServidorPeliculas
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt
.jar=49534:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile
.encoding=UTF-8 -classpath
/Users/sergiopalmerlopez/Desktop/PSP/ACT_Hilos_Sockets/out/production
/ACT_Hilos_Sockets Actividad.ServidorPeliculas
-----SERVIDOR-----
Servidor escuchando por puerto 12345
El servidor acepta la petición del cliente
Conexión establecida con el cliente Cliente_1
Consulta por título aceptada. Respuesta enviada
```

The "Client" tool window shows the interaction with the server:

```
Client
-----CLIENTE-----
CLIENTE: Esperando a que el servidor acepte la conexión
CLIENTE: Conexión establecida... a LocalHost por el puerto 12345
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
2
Consulta por título:
Escribe el título:
película 1
Consulta enviada al servidor
Cliente esperando respuesta del servidor...
CLIENTE: Servidor responde: [ServidorPeliculas{id=1, título='Película 1', nombreDirector='Director 1', precio=12.0}]
```

The "Run" tool window also shows the client's response:

```
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
|
```

At the bottom, the status bar indicates: "All files are up-to-date (moments ago)" and "113:34 LF UTF-8 4 spaces".

Aquí vemos la consulta por nombre del director.

```
ACT_Hilos_Sockets src Actividad ServidorPeliculas precio
Run: ServidorPeliculas
/ Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt
.jar=49534:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile
.encoding=UTF-8 -classpath
/Users/sergiopalmerlopez/Desktop/PSP/ACT_Hilos_Sockets/out/production
/ACT_Hilos_Sockets Actividad.ServidorPeliculas
-----SERVIDOR-----
Servidor escuchando por puerto 12345
El servidor acepta la petición del cliente
Conexión establecida con el cliente Cliente_1
Consulta por título aceptada. Respuesta enviada
Consulta por director aceptada. Respuesta enviada

-----CLIENTE-----
5. Salir:
-----
2 Consulta por título:
Escribe el título:
película 1
Consulta enviada al servidor
Cliente esperando respuesta del servidor...
CLIENTE: Servidor responde: [ServidorPeliculas{id=1, título='Pelicula 1', nombreDirector='Director 1', precio=12.0}]
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
3 Consulta por nombre del director:
Escribe el nombre del director:
director 5
Consulta enviada al servidor
Cliente esperando respuesta del servidor...
CLIENTE: Servidor responde: [ServidorPeliculas{id=5, título='Pelicula 5', nombreDirector='Director 5', precio=20.0}]
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
```

Aquí vemos cómo se añade una nueva película. Si todo ha salido bien, se le manda un mensaje al cliente.

The screenshot shows the IntelliJ IDEA interface with the project 'AE-1_Hilos_y_sockets' open. The 'client' module is selected, and the 'SocketCliente' run configuration is active. The terminal window titled 'SocketCliente' shows the following interaction:

```
BIENVENIDO A LA APLICACION CLIENTE
-----
CLIENTE: Esperando a que el servidor acepte la conexión
CLIENTE: Conexión establecida... a localhost por el puerto 8888
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
4
AÑADIENDO NUEVA PELÍCULA---
Escribe los datos para añadir una nueva película
ID:
34
Título:
Spiderman
Nombre del director:
Cristiano Ronaldo
Precio:
345.0
Consulta enviada al servidor
Cliente esperando respuesta del servidor...
CLIENTE: Servidor responde: La película se ha añadido con éxito:
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
```

The terminal also shows the Java command used to run the server:

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt
.jar=49294:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile
.encoding=UTF-8 -classpath
/Users/sergiopalmerolopez/Desktop/Hilos_Sockets_Christian/Hilos_y_sockets/AE
-1 Hilos_y_sockets/bin servidor.SocketServidor
```

At the bottom of the terminal, it says 'All files are up-to-date (a minute ago)' and '111:68 LF UTF-8 4 spaces'.

Aquí vemos como otro cliente se ha conectado al servidor y por lo tanto el cliente es el numero 2.

The screenshot shows the IntelliJ IDEA interface with the project `ACT_Hilos_Sockets` open. The left pane displays the code for `ServidorPeliculas.java`, which handles multiple client connections. The right pane shows the output of two terminal sessions: one for the server and one for a client. The server terminal shows it accepting connections from clients numbered 1 and 2. The client terminal shows the client connecting to the server and prompting for user input to choose a movie selection option.

```
Run: ServidorPelículas
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49522:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/sergiopalmerolopez/Desktop/PSP/ACT_Hilos_Sockets/out/production /ACT_Hilos_Sockets Actividad.ServidorPelículas
-----SERVIDOR-----
Servidor escuchando por puerto 12345
El servidor acepta la petición del cliente Cliente_1
Conexión establecida con el cliente Cliente_1
Consulta por id aceptada. Respuesta enviada
Cliente_1 ha cerrado la comunicación
El servidor acepta la petición del cliente
Conexión establecida con el cliente Cliente_2
|
```

```
Client: 
/Libary/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
-javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49530:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/sergiopalmerolopez/Desktop/PSP/ACT_Hilos_Sockets/out/production /ACT_Hilos_Sockets Actividad.Cliente
-----CLIENTE-----
CLIENTE: Esperando a que el servidor acepte la conexión
CLIENTE: Conexión establecida... a LocalHost por el puerto 12345
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
```

El servidor sigue recibiendo y escuchando peticiones del cliente hasta que este le mande la opción ‘5’, con la que se desconectará, pero este seguirá levantado a la espera de más clientes que se quieran conectar a él.

```
ACT_Hilos_Sockets src Actividad ServidorPeliculas precio
Project Client.java ServidorPeliculas.java Hilo.java
Run: ServidorPeliculas
Run: /library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
     -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=49522:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
     /Users/sergiopalmerlopez/Desktop/PSP/ACT_Hilos_Sockets/out/production/ACT_Hilos_Sockets Actividad.ServidorPeliculas
-----SERVIDOR-----
Servidor escuchando por puerto 12345
El servidor acepta la petición del cliente
Conexión establecida con el cliente Cliente_1
Consulta por id aceptada. Respuesta enviada
Cliente_1 ha cerrado la comunicación

-----CLIENTE-----
CLIENTE: Esperando a que el servidor acepte la conexión
CLIENTE: Conexión establecida... a LocalHost por el puerto 12345
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
1
Consulta por ID:
Escribe el id:
1
Consulta enviada al servidor
Cliente esperando respuesta del servidor...
CLIENTE: Servidor responde: ServidorPeliculas{id=1, titulo='Película 1', nombreDirector='Director 1', precio=12.0}
-----
Elige una opción:
1. Película por ID:
2. Película por Título:
3. Película por Nombre del director:
4. Añadir película:
5. Salir:
-----
5
Consulta enviada al servidor
Cliente esperando respuesta del servidor...
CLIENTE: Servidor responde: Fin del programa
Cliente: Fin del programa

Process finished with exit code 0
```