

CICLE FORMATIU: CFGS Desenvolupament Aplicacions Multiplataforma DAM

MÒDUL PROFESSIONAL: MP09 Programació de serveis i processos

UNITAT FORMATIVA: UF3. Sòcols i serveis

ACTIVITAT: Pt1. Programació de sòcols

---

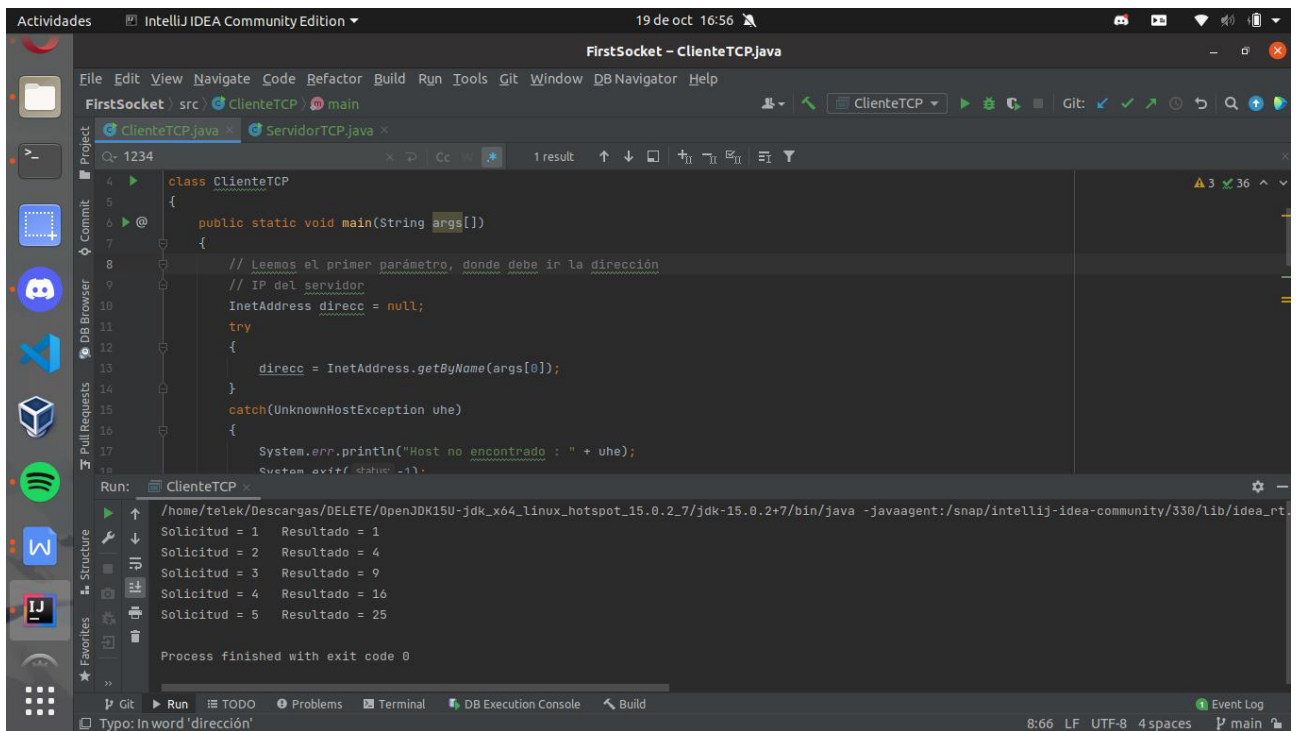
## **Pràctica 1. Programació de sòcols.**

En aquesta pràctica comprovarem com funcionen les comunicacions mitjançant sòcols en java, tant en TCP com en UDP.

- Carrega i prova els programes de clienteTCP.java i servidorTCP.java que trobaràs al document Exemples\_programacio\_sockets.pdf que hi ha al moodle.
- Carrega i prova també els programes de clienteUDP.java i servidorUDP.java que trobaràs al mateix document.
- Modifica els programes de manera que en comptes d'intercanviar-se un número i el seu quadrat, la comunicació sigui amb altres tipus de dades. Pots proposar alguna aplicació client-servidor senzilla. Explica detalladament el funcionament.
- Prova aquests programes amb el teu ordinador i un altre ordinador d'algun company (el programa client i el programa servidor en ordinadors diferents).
- Explica com funcionen les classes Socket, ServerSocket, DatagramSocket i DatagramPacket, i quan es fan servir.

### **Pràctica**

Entrega un document amb captures de pantalla, el programa modificat, i l'explicació de les classes de sockets.



The screenshot shows the IntelliJ IDEA Community Edition interface. The main editor displays the `ClienteTCP.java` file with the following code:

```
class ClienteTCP
{
    public static void main(String args[])
    {
        // Leemos el primer parámetro, donde debe ir la dirección
        // IP del servidor
        InetAddress direcc = null;
        try
        {
            direcc = InetAddress.getByName(args[0]);
        }
        catch (UnknownHostException uhe)
        {
            System.err.println("Host no encontrado : " + uhe);
            System.exit(1);
        }
    }
}
```

The Run window at the bottom shows the output of the program:

```
Solicitud = 1 Resultado = 1
Solicitud = 2 Resultado = 4
Solicitud = 3 Resultado = 9
Solicitud = 4 Resultado = 16
Solicitud = 5 Resultado = 25
Process finished with exit code 0
```

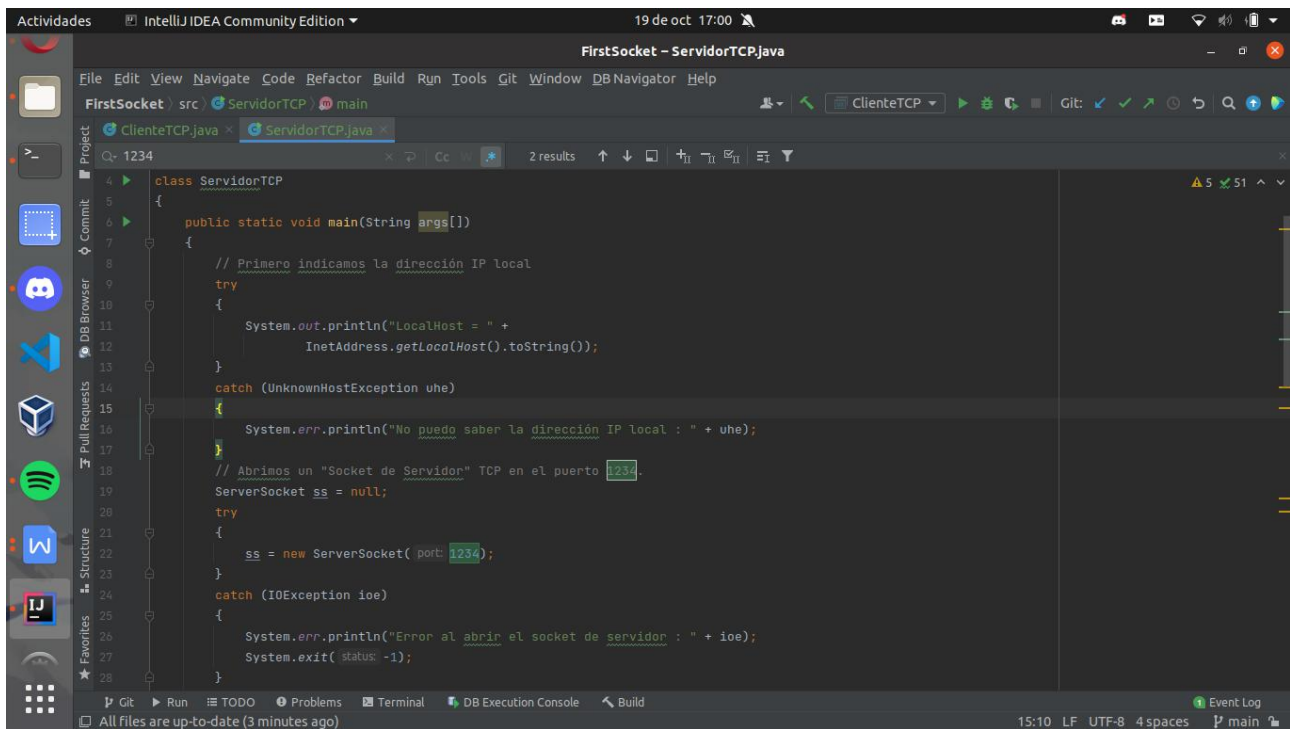
The status bar at the bottom indicates the file is in the `main` package, using the `UTF-8` encoding, and has 4 spaces for indentation.

El programa de `ClienteTCP` lo primero que hace es convertir el primer parámetro que se le pasa en una dirección de red, y esto lo consigue con la función `InetAddress.getByName()`. En caso de que no encuentre ningún host, muestra el error de host no encontrado y sale del programa.

En el caso de que encuentre el host, intenta conectarse al puerto que se le especifica, 1234 en este caso, y realiza una petición por cada parámetro extra que se le pasa. Cada uno de estos parámetros intenta convertirlos a número, y los manda al servidor para que realice una operación y le devuelva el resultado.

Para establecer la conexión, se declara una variable con el número del puerto, y luego otra variable de tipo `Socket` inicializada a `null`. Luego en un `try catch` se inicializa con un `new Socket(direcc, puerto)`, siendo `direcc` la dirección de host obtenida a partir del primer parámetro, y `puerto` es el que se ha especificado en la variable. Además de declarar un socket, se declaran dos variables más, una de `DataInputStream`, que recibe la respuesta del servidor, y `DataOutputStream`, que se encarga de enviar la información al servidor. Finalmente, una vez se han realizado las operaciones, se cierran ambos Data Streams. En caso de faltar el `try` en el que se encuentra todo esto, se muestra un mensaje de excepción.

Para acabar, el programa cliente intenta cerrar el socket de forma automática, y también gestiona la posibilidad de que haya un error al intentar hacer dicha operación.



```
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar
LocalHost = arch/192.168.122.1
Cliente = /192.168.0.189:40594   Entrada = 1 Salida = 1
Cliente = /192.168.0.189:40598   Entrada = 2 Salida = 4
Cliente = /192.168.0.189:40600   Entrada = 3 Salida = 9
Cliente = /192.168.0.189:40602   Entrada = 4 Salida = 16
Cliente = /192.168.0.189:40604   Entrada = 5 Salida = 25
```

El programa ServidorTCP, por su parte lo que hace es primero de todo indicar la dirección en localhost sobre la que va a funcionar, y después abrirá un nuevo socket en el puerto 1234. Una vez abierto el socket, esperará nuevas llamadas en bucle, y cuando llegue una la va a aceptar, creará un Input y Output DataStream, obtendrá el puerto del socket, la IP del cliente, y devolverá una respuesta al cliente mediante DataOutputStream.writeChars(salida), u otra función de tipo write. Finalmente, cerrará ambos DataStreams y cerrará la conexión con el cliente.

En caso de que falle cualquier cosa, siempre habrá un catch para gestionar la excepción.

El server socket funciona tal que cuando un cliente se conecta, el servidor abrirá un nuevo puerto para poder comunicarse exclusivamente con este.

El DatagramSocket es un servicio que se dedica exclusivamente a gestionar el envío y la recepción de paquetes en cada puerto del servidor. Por otra parte, DatagramPacket se utiliza más para conexiones UDP