Profesor: Nacho Mayorga

Correo electrónico: jimt66@educa.madrid.org

Programación de Servicios y Procesos

Análisis didáctico: Implementación básica de Cliente y Servidor TCP en Java con hilos.

Introducción

Este documento analiza un ejemplo básico de comunicación mediante sockets TCP en Java. Incluye la implementación de un cliente y un servidor que intercambian mensajes. Este tipo de comunicación es fundamental en el desarrollo de aplicaciones distribuidas.

Lo que sigue es un análisis crítico del código básico, que se da en la carpeta comprimida SocketBasicoTCPHilos.zip; después, se lista el código de los tres ficheros siguiendo las modificaciones sugeridas en dicho análisis inicial.

ServidorTCP.java

Resumen de funcionalidad

El servidor:

1. Escucha en el puerto 22222.
2. Espera conexiones de clientes.
3. Recibe un mensaje del cliente ("Hola ").
4. Responde con un mensaje fijo ("mundo!").
5. Cierra la conexión.

Estructura del código

1. ServerSocket ss = new ServerSocket(22222);
2. Creación del servidor:

```
ServerSocket ss = new ServerSocket(22222);
```

Esto inicializa el servidor para escuchar en el puerto especificado.

3. Bucle principal:

```
boolean continuar = true;
    BufferedReader br = new BufferedReader(new
    InputStreamReader(System.in));
    while (continuar) {
        Socket cliente = ss.accept();
        System.out.println("Servidor: Esperando mensaje...");
        BufferedReader clienteInput = new BufferedReader(new
    InputStreamReader(cliente.getInputStream()));
        String mensaje = clienteInput.readLine();
        if ("fin".equalsIgnoreCase(mensaje)) {
            continuar = false;
            System.out.println("Servidor: Recibido 'fin'. Terminando
    bucle.");
        }
    }
```

El servidor acepta conexiones en un bucle controlado. Cada conexión se gestiona secuencialmente, y el bucle finaliza cuando se recibe una cadena específica (por ejemplo, “fin”) como mensaje del cliente.

4. Lectura del mensaje del cliente:

```
BufferedReader br = new BufferedReader(new
InputStreamReader(cliente.getInputStream()));
String linea = br.readLine();
```

El servidor utiliza BufferedReader para leer datos enviados por el cliente.

5. Respuesta al cliente:

```
PrintWriter pw = new PrintWriter(cliente.getOutputStream());
pw.println("mundo!");
pw.flush();
```

Se usa PrintWriter para enviar un mensaje al cliente.

6. Cierre de recursos: Todos los flujos y el socket se cierran al final para liberar recursos.

Aspectos importantes:

- Puerto fijo: El servidor escucha en un puerto fijo (22222).
- Secuencialidad: Solo se atiende a un cliente a la vez.
- Uso de recursos: Cada conexión abre y cierra flujos dedicados.

ClienteTCP.java

Resumen de funcionalidad

El cliente:

1. Se conecta al servidor en 127.0.0.1:22222.
2. Envía un mensaje ("hola").
3. Recibe la respuesta del servidor ("mundo!").
4. Cierra la conexión.

Estructura del código

1. Conexión al servidor:

```
Socket s = new Socket("127.0.0.1", 22222);
```

Esto establece una conexión TCP con el servidor en la IP y puerto indicados.

2. Envío de mensaje:

```
PrintWriter pw = new PrintWriter(s.getOutputStream());
pw.println("hola");
pw.flush();
```

El cliente envía el mensaje utilizando un flujo de escritura.

3. Recepción de respuesta:

```
BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
String linea = br.readLine();
System.out.println(linea);
```

   Se utiliza un flujo de lectura para recibir la respuesta del servidor.

4. Cierre de recursos: Los flujos y el socket se cierran al finalizar.

Aspectos importantes

- Dirección fija: El cliente utiliza la dirección local (127.0.0.1) para conectarse al servidor.
- Simplicidad: Envía y recibe un único mensaje por conexión.

Ventajas y Limitaciones del Código

Ventajas

1. Simplicidad: El código es directo y adecuado para introducir conceptos básicos de sockets.
2. Reutilizable: Se puede ampliar para manejar múltiples clientes o mejorar la funcionalidad.

Limitaciones

1. Secuencialidad: El servidor no puede manejar múltiples clientes simultáneamente.
2. E/S básica: Solo se gestionan cadenas de texto simples.
3. Falta de robustez: No incluye manejo avanzado de errores ni tiempo de espera (timeouts).

Propuestas de Ampliación

1. Servidor multihilo:
   - Implementar un hilo por conexión para manejar múltiples clientes simultáneamente.
2. Mensajes más complejos:
   - Permitir el intercambio de estructuras más complejas (JSON, objetos Java serializados).
3. Manejo de errores avanzado:
   - Agregar control de excepciones para mejorar la robustez.

Anexo: Código Propuesto para Mejoras

ServidorTCP.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class ServidorTCP {

    public static void main(String[] args) {
        try (ServerSocket ss = new ServerSocket(22222)) {
            System.out.println("Servidor iniciado en el puerto 22222");
            boolean continuar = true;

            while (continuar) {
                Socket cliente = ss.accept();
                System.out.println("Cliente conectado");

                try (BufferedReader clienteInput = new BufferedReader(new
InputStreamReader(cliente.getInputStream()));
                     PrintWriter clienteOutput = new
PrintWriter(cliente.getOutputStream(), true)) {

                    String mensaje = clienteInput.readLine();
                    if ("fin".equalsIgnoreCase(mensaje)) {
                        continuar = false;
                        System.out.println("Servidor: Recibido 'fin'.
Terminando...");
                    } else {
                        clienteOutput.println("mundo!");
                    }
```

```
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ClienteTCP.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class ClienteTCP {

    public static void main(String[] args) {
        try (Socket socket = new Socket("127.0.0.1", 22222);
             PrintWriter output = new PrintWriter(socket.getOutputStream(),
true);
             BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()))) {

            output.println("hola");
            String respuesta = input.readLine();
            System.out.println("Respuesta del servidor: " + respuesta);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```