UD3: Programación de comunicaciones en red

Práctica 1. Entendiendo los sockets

Programación de Servicios y Procesos

Helena Brau Bou

Curso 2º DAM 2025/26

Índice

Ejercicio 1: Ejecución del código..... 2
Ejecuta el código del cliente y después el del servidor. ¿Qué ocurre? ¿Por qué?..... 2
¿Qué protocolo de comunicación se está utilizando UDP o TCP?..... 2
Ejecuta el servidor y varias instancias de cliente, utiliza el chat y haz captura.....2
Ejercicio 2: Comprender el Código del Cliente..... 3
¿Cuál es el propósito de la clase ClienteChat en este código?..... 3
Explica el flujo de comunicación entre el cliente y el servidor. ¿Cómo se envían y reciben los mensajes?..... 3
¿Qué hace el método ejecutar() en la clase ClienteChat?.....3
¿Qué sucede si el cliente intenta enviar un mensaje después de que se ha desconectado del servidor?..... 3
Ejercicio 3: Comprender el Código del Servidor..... 4
¿Cuál es el papel de la clase HiloServidor en el servidor?..... 4
Explica el propósito de la matriz tabla en la clase ServidorChat. ¿Cómo se utiliza para controlar las conexiones?.....5
Modifica el código del servidor para permitir un número máximo diferente de conexiones simultáneas. ¿Qué cambios debes hacer en el código?.....5
¿Qué sucede si un cliente se desconecta abruptamente sin salir adecuadamente?..... 5
Ejercicio 4: Escenarios de Error y Manejo de Excepciones..... 6
¿Qué sucede si el servidor se cierra inesperadamente mientras hay clientes conectados?..... 6
¿Qué excepciones maneja el código del cliente y del servidor? ¿Cómo se manejan estas excepciones?..... 7

Ejercicio 1: Ejecución del código

Ejecuta el código del cliente y después el del servidor. ¿Qué ocurre? ¿Por qué?

Si ejecuto primero el cliente, el cliente intenta conectarse al servidor en localhost puerto 44444 (`s = new Socket("localhost", puerto);`). Como el servidor aún no está ejecutándose, el cliente no puede conectarse y muestra un mensaje de error: "IMPOSIBLE CONECTAR CON EL SERVIDOR"

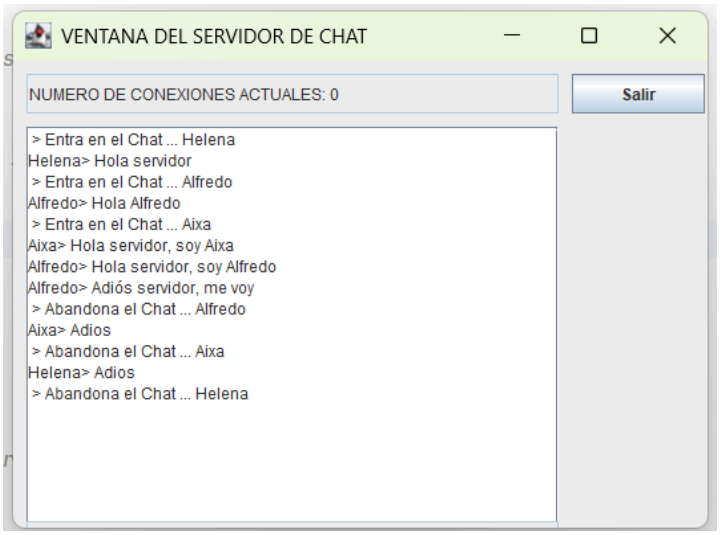
Si ejecuto primero el servidor y luego el cliente, el servidor abre un ServerSocket en el puerto 44444 y empieza a escuchar conexiones. Cuando el cliente se conecta, el servidor acepta la conexión, se crea un hilo (HiloServidor) y el chat puede empezar a enviar mensajes.

¿Qué protocolo de comunicación se está utilizando UDP o TCP?

Se está utilizando en protocolo de comunicación TCP porque el código utiliza Socket y ServerSocket. Estos objetos pertenecen a la API de TCP de Java, no a UDP.

Ejecuta el servidor y varias instancias de cliente, utiliza el chat y haz captura.

1. Ejecuto el servidor (ServidorChat).
2. Ejecuto varias instancias del cliente (ClienteChat)
3. Cada cliente introduce su nombre (nick).
4. Todos los clientes que se conectan pueden enviar mensajes, y el servidor mantiene el historial, mostrando los mensajes en tiempo real a todos los clientes.



Ejercicio 2: Comprender el Código del Cliente

¿Cuál es el propósito de la clase ClienteChat en este código?

ClienteChat es la clase que representa a un cliente en el chat. Su objetivo principal es permitir que un usuario se conecte a un servidor, envíe mensajes, reciba mensajes de otros clientes y vea todo en una interfaz gráfica.

Explica el flujo de comunicación entre el cliente y el servidor. ¿Cómo se envían y reciben los mensajes?

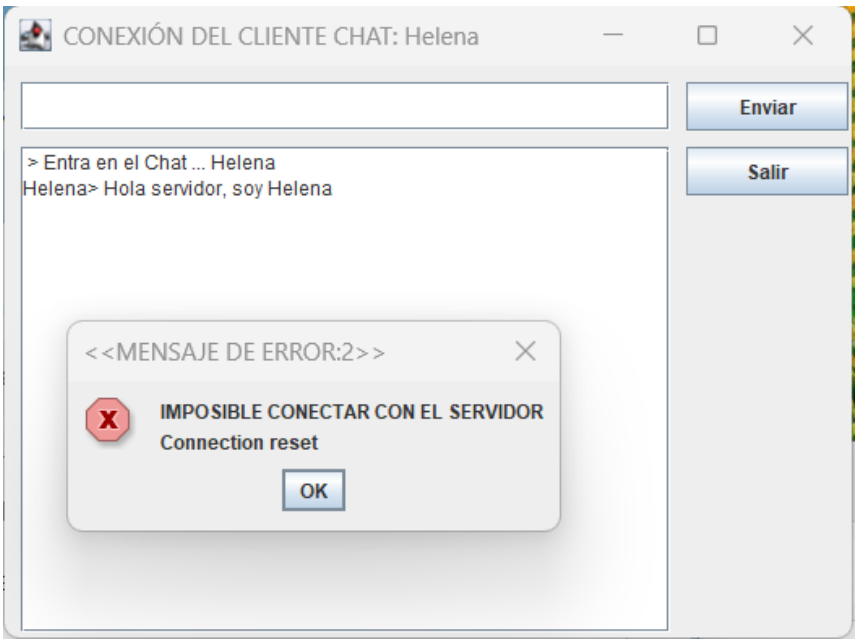
1. Conexión al servidor
    - El cliente se conecta al servidor usando el socket en localhost:44444.
    - Crea flujos de entrada y salida para enviar y recibir mensajes.
    - Envía un mensaje inicial al servidor indicando que el cliente ha entrado al chat.
2. Recibir mensajes
    - El método ejecutar() mantiene un bucle que lee continuamente mensajes desde el servidor con fentrada.readUTF().
    - Cada mensaje recibido se muestra en textarea1.
3. Enviar mensajes
    - El usuario escribe un mensaje en mensaje y pulsa “Enviar”.
    - Se construye un texto con el nombre del cliente y se envía al servidor mediante fsalida.writeUTF(texto).
4. Desconexión
    - Al pulsar “Salir”, el cliente envía un mensaje especial "*" al servidor para indicar que se desconecta.
    - Cambia repetir = false para salir del bucle de lectura de mensajes.
5. Cierre del cliente
    - Cuando el cliente se desconecta o el servidor cierra la conexión, se cierra el socket y termina el programa.

¿Qué hace el método ejecutar() en la clase ClienteChat?

El método ejecutar() es el encargado de escuchar continuamente los mensajes que llegan desde el servidor y mostrarlos en la interfaz gráfica del cliente.

¿Qué sucede si el cliente intenta enviar un mensaje después de que se ha desconectado del servidor?

En ClienteChat, el método ejecutar() está constantemente leyendo del servidor con fentrada.readUTF(). Cuando el servidor se desconecta, se lanza una IOException. Esto provoca que se muestre inmediatamente el mensaje de error: IMPOSIBLE CONECTAR CON EL SERVIDOR.



Además, repetir se pone en false, por lo que el bucle de lectura termina.

El cliente ya no puede interactuar, ni enviar mensajes, ni pulsar el botón “Enviar”, porque la aplicación se queda inactiva inmediatamente después de detectar la desconexión del servidor.

Ejercicio 3: Comprender el Código del Servidor

¿Cuál es el papel de la clase HiloServidor en el servidor?

Cuando un cliente se conecta se crea un hilo independiente (HiloServidor). La clase HiloServidor lee continuamente los mensajes del cliente, detecta cuando se desconecta, actualiza el número de clientes activos y reenvía todos los mensajes a los demás clientes para mantenerlos sincronizados.

Explica el propósito de la matriz tabla en la clase ServidorChat. ¿Cómo se utiliza para controlar las conexiones?

La matriz tabla en la clase ServidorChat sirve para almacenar los sockets de todos los clientes conectados. Su propósito es mantener un registro de todos los clientes activos para poder enviarles mensajes. A la vez, permite que el servidor tenga acceso a cada cliente individualmente para gestionar la comunicación.

Se utiliza de la siguiente manera: Cada vez que un cliente se conecta, el socket que representa esa conexión se guarda en tabla[CONEXIONES] y se incrementan los contadores CONEXIONES y ACTUALES. Cuando el servidor necesita reenviar mensajes a todos los clientes, recorre la matriz tabla y usa cada socket para enviar el texto mediante DataOutputStream.

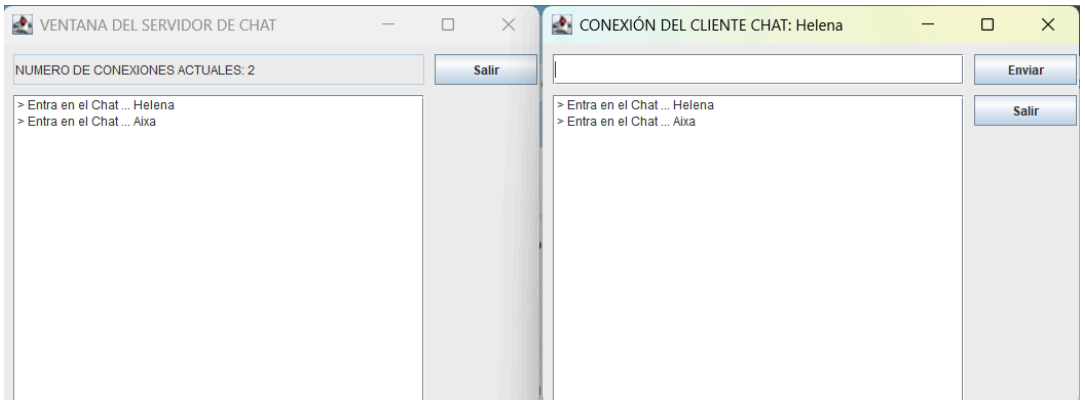
Modifica el código del servidor para permitir un número máximo diferente de conexiones simultáneas. ¿Qué cambios debes hacer en el código?

El código actual tiene un máximo de 4 conexiones simultáneas permitidas pero una tabla que tiene capacidad para 10 clientes. Si queremos permitir un máximo de 10 conexiones solo tenemos que cambiar el máximo a 10 (static int MAXIMO=10). Si quisierámos permitir, por ejemplo, 15 conexiones simultáneas, tendríamos que cambiar el máximo a, por ejemplo, 15 (static int MAXIMO=15) y cambiar la capacidad de la tabla también a 15 (tabla = new Socket[15]).

¿Qué sucede si un cliente se desconecta abruptamente sin salir adecuadamente?

Si un cliente se desconecta abruptamente, el HiloServidor intenta leer del DataInputStream (fentrada.readUTF()), detecta un error de E/S, el bloque catch captura la excepción (SocketException) y el while del hilo se rompe, finalizando el hilo de ese cliente.

Además, como el cliente no envió el mensaje especial "*", no se actualiza el número de conexiones y el resto de clientes tampoco reciben un mensaje conforme uno de los clientes se ha desconectado. En la captura siguiente se muestra que se han conectado dos clientes. Hemos cerrado al cliente Aixa abruptamente pero se sigue mostrando que el número de conexiones actuales es 2 y el cliente no ha recibido ningún mensaje informando de la desconexión del otro cliente.

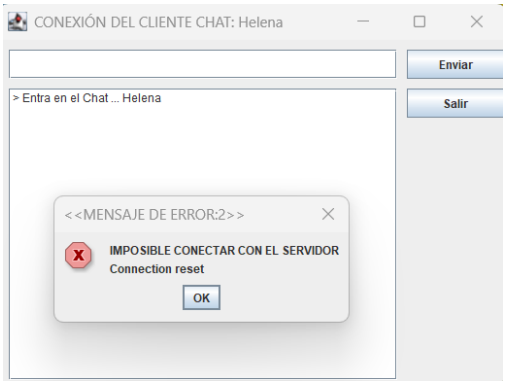


Ejercicio 4: Escenarios de Error y Manejo de Excepciones

¿Qué sucede si el servidor se cierra inesperadamente mientras hay clientes conectados?

Para los clientes conectados: Cada cliente tiene un DataInputStream abierto (texto = fentrada.readUTF()) esperando mensajes del servidor. Cuando el servidor se cierra, el socket del cliente, a través del método ejecutar() de ClienteChat, detecta que la conexión se ha roto, lanza una IOException mostrando el mensaje de error “IMPOSIBLE CONECTAR CON EL SERVIDOR”, la variable repetir se pone a false, terminando así el bucle de lectura de mensajes. Finalmente, el socket se cierra y el cliente termina.

Para el servidor: Los hilos de los clientes (HiloServidor) no tienen más conexión con el cliente y cualquier intento de leer/escribir desde esos sockets termina en excepción. Como el servidor se cerró abruptamente, los contadores (ACTUALES, CONEXIONES) no se actualizan automáticamente.



¿Qué excepciones maneja el código del cliente y del servidor? ¿Cómo se manejan estas excepciones?

El código del **cliente (ClienteChat)** maneja las siguientes excepciones:

1. **IOException al crear los flujos o enviar/recibir mensajes:** Captura errores al conectarse o enviar datos al servidor, y se maneja mostrando un mensaje de error, imprimiendo el stacktrace y cerrando el programa con System.exit(0).

```
try {
    fentrada = new DataInputStream(socket.getInputStream()); // Crear flujo de entrada al socket
    fsalida = new DataOutputStream(socket.getOutputStream()); // CREO FLUJO DE SALIDA AL socket de escritura

    // Mensaje inicial indicando que el usuario ha entrado al chat
    String texto = " > Entra en el Chat ... " + nombre;
    fsalida.writeUTF(texto);
} catch (IOException e) {
    System.out.println("ERROR DE E/S");
    e.printStackTrace();
    System.exit( status: 0);
}
```

2. **IOException al leer mensajes del servidor en el método ejecutar():** Captura que el servidor se ha cerrado o la conexión se ha perdido, y se maneja mostrando un cuadro de error, poniendo repetir = false para terminar el bucle y cerrando el socket del cliente.
   Ejemplo de un cliente intentando conectarse sin haber iniciado el servidor:

```
141         try {
142             s = new Socket( host: "localhost", puerto); // conectarse al servidor   s: null    puerto: 44444
            } catch (IOException e) {   e: "java.net.ConnectException: Connection refused: connect"
144             JOptionPane.showMessageDialog( parentComponent: null,
145                     message: "IMPOSIBLE CONECTAR CON EL SERVIDOR\n" + e.getMessage(),
146                     title: "<<MENSAJE DE ERROR:1>>", JOptionPane.ERROR_MESSAGE);
147             System.exit( status: 0) [Method will fail] ;
148         }
149         if (!nombre.trim().equals("")) {
150             ClienteChat cliente = new ClienteChat(s, nombre);
151             cliente.setBounds( x: 0,  y: 0,  width: 540,  height: 400);
152             cliente.setVisible(true);
153             cliente.ejecutar(); // iniciar bucle de lectura de mensajes
154         } else {
155             System.out.println("El nombre está vacio....");
156         }
```

```
Debug   ServidorChat   ClienteChat
Threads & Variables   Console
"main"@1 in g...ain": RUNNING
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
main:144, ClienteChat (cliente)
static members of ClienteChat
args = {String[0]@1942} []
puerto = 44444
nombre = "Helena"
s = null
e = {ConnectException@1944} "java.net.ConnectException: Connection refused: connect"
```

El **servidor (HiloServidor y ServidorChat)** maneja las siguientes excepciones:

1. **IOException / SocketException al leer mensajes de los clientes:** Captura que un cliente se ha desconectado de forma abrupta, y se maneja imprimiendo la excepción y saliendo del bucle para finalizar el hilo del cliente.

```
    // Bucle infinito para recibir mensajes continuamente
    while (true) {
        String cadena = "";
        try {
            cadena = fentrada.readUTF(); // leemos mensaje del cliente
            if (cadena.trim().equals("*")) {// si el mensaje es "*", el cliente se desconecta
                ServidorChat.ACTUALES--; // actualizamos el contador de clientes
                ServidorChat.mensaje
                        .setText("NUMERO DE CONEXIONES ACTUALES: "
                                + ServidorChat.ACTUALES);// Datos.getConexiones());
                break; // salimos del bucle y terminamos el hilo
            }

            // Añadimos el mensaje recibido al historial del servidor
            ServidorChat.textarea.append(cadena + "\n");

            // Obtenemos el historial completo y lo enviamos a todos los clientes
            texto = ServidorChat.textarea.getText();
            EnviarMensajes(texto);

        } catch (Exception e) {
            e.printStackTrace(); // en caso de error, imprimimos y salimos
            break;
        }
    }// fin while
}//run
```

2. **SocketException / IOException al enviar mensajes a los clientes:** Captura que un cliente se desconecta mientras se envían mensajes, y se maneja mostrando un mensaje en consola sin detener el servidor ni afectar a los demás clientes.

```
// Metodo que envía el historial de mensajes a todos los clientes conectados
private void EnviarMensajes(String texto) {  2 usages
    int i;
    //recorremos tabla de sockets de todos los clientes para enviarles los mensajes
    for (i = 0; i < ServidorChat.CONEXIONES; i++) {
        Socket s1;
        s1 = ServidorChat.tabla[i];
        try {
            // Creamos un flujo de salida para cada cliente y enviamos el mensaje
            DataOutputStream fsalida2 = new DataOutputStream(
                    s1.getOutputStream());
            fsalida2.writeUTF(texto);
        } catch (SocketException se) {//hay que dejarlo
            // puede ocurrir cuando finalizo cliente
            // sale cuando un cliente ha finalizado
            System.out.println(" SOCKET EXCEPTION 2 : " + se.getMessage());
        } catch (IOException e) {
            e.printStackTrace();
            //System.out.println(" IO EXCEPTION  : " + e.getMessage());
        }
    }//for
}//EnviarMensajes
```

3. **IOException al aceptar conexiones en ServidorChat:** Captura que el servidor ha sido cerrado mientras esperaba conexiones, y se maneja rompiendo el bucle de aceptación para proceder a cerrar el servidor.

```
// Bucle que acepta conexiones mientras no se alcance el máximo permitido
while (CONEXIONES < MAXIMO) {
    Socket s = new Socket();
    try {
        s = servidor.accept();// el servidor se queda esperando un cliente
    } catch (SocketException ns) {
        // Si pulsamos salir, el servidor se cierra y accept() lanza excepción
        break;
    }

    // Guardamos el socket del cliente en la tabla
    tabla[CONEXIONES] = s;

    // Actualizamos contadores
    CONEXIONES++;
    ACTUALES++;

    // Cada cliente tendrá un hilo independiente que lo atiende
    HiloServidor hilo = new HiloServidor(s);
    hilo.start();
}
```