### **EJERCICIO EXAMEN**

la clase HiloServdior se encarga de recoger la consulta del cliente en concreto y mandar su respuesta

#### **tcp**

import java.io.Serializable;

public class Asignatura implements Serializable {

private int id;

private String nombre;

public Asignatura(int id, String nombre) {

this.id = id;

this.nombre = nombre;

}

public int getId() { return id; }

public String getNombre() { return nombre; }

@Override

public String toString() {

return "Asignatura: " + id + " - " + nombre;

}

}

import java.io.Serializable;

public class Especialidad implements Serializable {

private int id;

private String nombre;

public Especialidad(int id, String nombre) {

this.id = id;

this.nombre = nombre;

}

public int getId() { return id; }

public String getNombre() { return nombre; }

@Override

public String toString() {

return "Especialidad: " + id + " - " + nombre;

}

}

import java.io.Serializable;

public class Profesor implements Serializable {

private int id;

private String nombre;

private Especialidad especialidad;

private Asignatura[] asignaturas;

public Profesor(int id, String nombre, Especialidad especialidad, Asignatura[] asignaturas) {

this.id = id;

this.nombre = nombre;

this.especialidad = especialidad;

this.asignaturas = asignaturas;

}

public int getId() { return id; }

public String getNombre() { return nombre; }

public Especialidad getEspecialidad() { return especialidad; }

public Asignatura[] getAsignaturas() { return asignaturas; }

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append("Nombre: ").append(nombre).append("\n");

sb.append(especialidad.toString()).append("\n");

for (Asignatura a : asignaturas) {

sb.append(a.toString()).append("\n");

}

return sb.toString();

}

}

import java.io.\*;

import java.net.\*;

public class ServidorTCP {

private static final int PUERTO = 44444;

private static Profesor[] profesores = new Profesor[3]; // Solo 3 profesores en el array

private static int contadorClientes = 1;

public static void main(String[] args) {

try (ServerSocket servidor = new ServerSocket(PUERTO)) {

System.out.println("Servidor iniciado...");

// Definir profesores manualmente

Especialidad esp = new Especialidad(1, "INFORMÁTICA");

profesores[0] = new Profesor(1, "Antonio", esp, new Asignatura[]{

new Asignatura(2, "ADAT"),

new Asignatura(3, "PSP"),

new Asignatura(4, "PMD")

});

profesores[1] = new Profesor(2, "María", esp, new Asignatura[]{

new Asignatura(5, "BBDD"),

new Asignatura(6, "Sistemas Operativos"),

new Asignatura(7, "Redes")

});

profesores[2] = new Profesor(3, "José", esp, new Asignatura[]{

new Asignatura(8, "Matemáticas"),

new Asignatura(9, "Física"),

new Asignatura(10, "Química")

});

while (true) {

Socket cliente = servidor.accept();

System.out.println("Cliente " + contadorClientes + " conectado");

new HiloServidor(cliente, contadorClientes).start();

contadorClientes++;

}

} catch (IOException e) {

e.printStackTrace();

}

}

// Método para buscar un profesor por ID

public static Profesor buscarProfesor(int id) {

for (Profesor p : profesores) {

if (p != null && p.getId() == id) {

return p;

}

}

return new Profesor(-1, "No encontrado", new Especialidad(0, "N/A"), new Asignatura[0]);

}

}

class HiloServidor extends Thread {

private Socket socket;

private int idCliente;

public HiloServidor(Socket socket, int idCliente) {

this.socket = socket;

this.idCliente = idCliente;

}

public void run() {

try (ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {

out.writeInt(idCliente);

out.flush();

while (true) {

int idProfesor = in.readInt();

System.out.println("Cliente " + idCliente + " consultando id: " + idProfesor);

Profesor profesor = ServidorTCP.buscarProfesor(idProfesor);

out.writeObject(profesor);

out.flush();

}

} catch (IOException e) {

System.out.println("Cliente " + idCliente + " desconectado.");

}

}

}

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class ClienteTCP {

public static void main(String[] args) {

try (Socket socket = new Socket("localhost", 44444);

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

Scanner scanner = new Scanner(System.in)) {

int idCliente = in.readInt();

System.out.println("Soy el cliente: " + idCliente);

while (true) {

System.out.print("Introduce identificador a consultar: ");

int idProfesor = scanner.nextInt();

out.writeInt(idProfesor);

out.flush();

Profesor profesor = (Profesor) in.readObject();

System.out.println(profesor.toString());

}

} catch (IOException | ClassNotFoundException e) {

System.out.println("Desconectado del servidor.");

}

}

}

#### **udp**

import java.io.\*;

import java.net.\*;

public class ServidorUDP {

private static final int PUERTO = 44444;

private static Profesor[] profesores = new Profesor[3];

public static void main(String[] args) {

try (DatagramSocket socket = new DatagramSocket(PUERTO)) {

System.out.println("Servidor UDP iniciado...");

// Inicializar profesores

Especialidad esp = new Especialidad(1, "INFORMÁTICA");

profesores[0] = new Profesor(1, "Antonio", esp, new Asignatura[]{

new Asignatura(2, "ADAT"),

new Asignatura(3, "PSP"),

new Asignatura(4, "PMD")

});

byte[] buffer = new byte[1024];

while (true) {

DatagramPacket paquete = new DatagramPacket(buffer, buffer.length);

socket.receive(paquete);

int idProfesor = Integer.parseInt(new String(paquete.getData()).trim());

Profesor profesor = buscarProfesor(idProfesor);

ByteArrayOutputStream bos = new ByteArrayOutputStream();

ObjectOutputStream oos = new ObjectOutputStream(bos);

oos.writeObject(profesor);

oos.flush();

byte[] respuesta = bos.toByteArray();

DatagramPacket paqueteRespuesta = new DatagramPacket(respuesta, respuesta.length, paquete.getAddress(), paquete.getPort());

socket.send(paqueteRespuesta);

}

} catch (IOException e) {

e.printStackTrace();

}

}

public static Profesor buscarProfesor(int id) {

for (Profesor p : profesores) {

if (p != null && p.getId() == id) {

return p;

}

}

return new Profesor(-1, "No encontrado", new Especialidad(0, "N/A"), new Asignatura[0]);

}

}

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class ClienteUDP {

public static void main(String[] args) {

final String SERVIDOR = "localhost"; // Cambia esto si el servidor está en otra IP

final int PUERTO\_SERVIDOR = 44444;

Scanner scanner = new Scanner(System.in);

try (DatagramSocket socket = new DatagramSocket()) {

while (true) {

// Pedir al usuario el ID del profesor a consultar

System.out.print("Introduce el ID del profesor a consultar: ");

String mensaje = scanner.nextLine();

// Enviar ID del profesor al servidor

byte[] bufferEnvio = mensaje.getBytes();

InetAddress direccionServidor = InetAddress.getByName(SERVIDOR);

DatagramPacket paqueteEnvio = new DatagramPacket(bufferEnvio, bufferEnvio.length, direccionServidor, PUERTO\_SERVIDOR);

socket.send(paqueteEnvio);

// Recibir respuesta del servidor (objeto Profesor serializado)

byte[] bufferRecepcion = new byte[4096]; // Buffer grande para recibir objetos

DatagramPacket paqueteRecepcion = new DatagramPacket(bufferRecepcion, bufferRecepcion.length);

socket.receive(paqueteRecepcion);

// Convertir los bytes en un objeto Profesor

ByteArrayInputStream bis = new ByteArrayInputStream(paqueteRecepcion.getData());

ObjectInputStream ois = new ObjectInputStream(bis);

Profesor profesor = (Profesor) ois.readObject();

// Mostrar el resultado en pantalla

System.out.println("\nRespuesta del servidor:");

System.out.println(profesor.toString());

}

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

### **CHAT SERVIDOR CLIENTE**

**El servidor acepta conexiones de múltiples clientes.**

**Cada cliente puede enviar mensajes al servidor.**

**El servidor retransmite los mensajes a todos los clientes conectados.**

**Si un cliente envía \*, se desconecta.**

#### **TCP**

import java.io.\*;

import java.net.\*;

public class ServidorChat {

static final int MAXIMO = 10; // Máximo de conexiones permitidas

static final int PUERTO = 44444;

static ServerSocket servidor;

public static void main(String[] args) throws IOException {

System.out.println("Servidor iniciado en el puerto " + PUERTO);

servidor = new ServerSocket(PUERTO);

Socket tabla[] = new Socket[MAXIMO]; // Almacena los sockets de los clientes

ComunHilos comun = new ComunHilos(MAXIMO, 0, 0, tabla); // Objeto compartido entre los hilos

while (comun.getCONEXIONES() < MAXIMO) {

Socket socket = servidor.accept(); // Esperando conexión de cliente

comun.addTabla(socket, comun.getCONEXIONES());

comun.setACTUALES(comun.getACTUALES() + 1);

comun.setCONEXIONES(comun.getCONEXIONES() + 1);

HiloServidorChat hilo = new HiloServidorChat(socket, comun);

hilo.start();

}

servidor.close();

}

}

import java.io.\*;

import java.net.\*;

public class HiloServidorChat extends Thread {

DataInputStream fentrada;

Socket socket;

ComunHilos comun;

public HiloServidorChat(Socket s, ComunHilos comun) {

this.socket = s;

this.comun = comun;

try {

fentrada = new DataInputStream(socket.getInputStream());

} catch (IOException e) {

System.out.println("ERROR DE E/S");

e.printStackTrace();

}

}

public void run() {

System.out.println("NÚMERO DE CONEXIONES ACTUALES: " + comun.getACTUALES());

// Enviar todos los mensajes previos al cliente recién conectado

String texto = comun.getMensajes();

EnviarMensajesaTodos(texto);

while (true) {

try {

String cadena = fentrada.readUTF();

if (cadena.trim().equals("\*")) {

comun.setACTUALES(comun.getACTUALES() - 1);

System.out.println("NÚMERO DE CONEXIONES ACTUALES: " + comun.getACTUALES());

break;

}

comun.setMensajes(comun.getMensajes() + cadena + "\n");

EnviarMensajesaTodos(comun.getMensajes());

} catch (IOException e) {

e.printStackTrace();

break;

}

}

try {

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private void EnviarMensajesaTodos(String texto) {

for (int i = 0; i < comun.getCONEXIONES(); i++) {

Socket s1 = comun.getElementoTabla(i);

if (!s1.isClosed()) {

try {

DataOutputStream fsalida2 = new DataOutputStream(s1.getOutputStream());

fsalida2.writeUTF(texto);

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

import java.io.\*;

import java.net.\*;

import javax.swing.\*;

public class ClienteChat extends JFrame implements Runnable {

static final int PUERTO = 44444;

Socket socket;

DataInputStream fentrada;

DataOutputStream fsalida;

JTextArea textArea;

JTextField mensajeField;

JButton enviarButton, salirButton;

boolean repetir = true;

String nombre;

public ClienteChat(Socket s, String nombre) {

this.socket = s;

this.nombre = nombre;

setTitle("CONEXIÓN DEL CLIENTE CHAT: " + nombre);

setSize(500, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BoxLayout(getContentPane(), BoxLayout.Y\_AXIS));

textArea = new JTextArea();

textArea.setEditable(false);

add(new JScrollPane(textArea));

mensajeField = new JTextField();

add(mensajeField);

enviarButton = new JButton("Enviar");

salirButton = new JButton("Salir");

JPanel panelBotones = new JPanel();

panelBotones.add(enviarButton);

panelBotones.add(salirButton);

add(panelBotones);

enviarButton.addActionListener(e -> enviarMensaje());

salirButton.addActionListener(e -> salirChat());

setVisible(true);

try {

fentrada = new DataInputStream(socket.getInputStream());

fsalida = new DataOutputStream(socket.getOutputStream());

fsalida.writeUTF(nombre + " ha entrado al chat.");

} catch (IOException e) {

e.printStackTrace();

}

}

public void enviarMensaje() {

try {

String mensaje = nombre + ": " + mensajeField.getText();

fsalida.writeUTF(mensaje);

mensajeField.setText("");

} catch (IOException e) {

e.printStackTrace();

}

}

public void salirChat() {

try {

fsalida.writeUTF("\*");

repetir = false;

socket.close();

System.exit(0);

} catch (IOException e) {

e.printStackTrace();

}

}

public void run() {

while (repetir) {

try {

String texto = fentrada.readUTF();

textArea.setText(texto);

} catch (IOException e) {

JOptionPane.showMessageDialog(null, "IMPOSIBLE CONECTAR CON EL SERVIDOR\n" + e.getMessage(),

"MENSAJE DE ERROR", JOptionPane.ERROR\_MESSAGE);

repetir = false;

}

}

}

public static void main(String[] args) {

try {

String nombre = JOptionPane.showInputDialog("Introduce tu nombre:");

if (nombre.trim().length() == 0) {

System.out.println("El nombre está vacío...");

return;

}

Socket s = new Socket("localhost", PUERTO);

ClienteChat cliente = new ClienteChat(s, nombre);

new Thread(cliente).start();

} catch (IOException e) {

JOptionPane.showMessageDialog(null, "IMPOSIBLE CONECTAR CON EL SERVIDOR\n" + e.getMessage(),

"MENSAJE DE ERROR", JOptionPane.ERROR\_MESSAGE);

}

}

}

#### **UDP**

import java.net.\*;

public class ServidorUDP {

public static void main(String[] args) {

final int PUERTO = 12345;

final String MULTICAST\_IP = "225.0.0.1";

byte[] buffer = new byte[1024];

try {

MulticastSocket socket = new MulticastSocket(PUERTO);

InetAddress grupo = InetAddress.getByName(MULTICAST\_IP);

socket.joinGroup(grupo);

System.out.println("Servidor UDP en ejecución...");

while (true) {

// Recibir mensaje del cliente

DatagramPacket paquete = new DatagramPacket(buffer, buffer.length);

socket.receive(paquete);

String mensaje = new String(paquete.getData(), 0, paquete.getLength());

System.out.println("Mensaje recibido: " + mensaje);

// Reenviar el mensaje a todos los clientes en el grupo multicast

DatagramPacket paqueteEnvio = new DatagramPacket(mensaje.getBytes(), mensaje.length(), grupo, PUERTO);

socket.send(paqueteEnvio);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

import java.awt.\*;

import java.awt.event.\*;

import java.net.\*;

import javax.swing.\*;

public class ClienteUDP extends JFrame implements ActionListener, Runnable {

private JTextArea chatArea;

private JTextField mensajeField;

private JButton enviarBtn, salirBtn;

private MulticastSocket socket;

private InetAddress grupo;

private final int PUERTO = 12345;

private final String MULTICAST\_IP = "225.0.0.1";

private String nombre;

public ClienteUDP(String nombre) {

this.nombre = nombre;

// Configurar la interfaz gráfica

setTitle("Chat UDP - " + nombre);

setSize(500, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

chatArea = new JTextArea();

chatArea.setEditable(false);

add(new JScrollPane(chatArea), BorderLayout.CENTER);

JPanel panel = new JPanel();

panel.setLayout(new BorderLayout());

mensajeField = new JTextField();

panel.add(mensajeField, BorderLayout.CENTER);

enviarBtn = new JButton("Enviar");

enviarBtn.addActionListener(this);

panel.add(enviarBtn, BorderLayout.EAST);

salirBtn = new JButton("Salir");

salirBtn.addActionListener(this);

add(salirBtn, BorderLayout.SOUTH);

add(panel, BorderLayout.NORTH);

try {

socket = new MulticastSocket(PUERTO);

grupo = InetAddress.getByName(MULTICAST\_IP);

socket.joinGroup(grupo);

} catch (Exception e) {

e.printStackTrace();

}

new Thread(this).start();

}

@Override

public void actionPerformed(ActionEvent e) {

try {

if (e.getSource() == enviarBtn) {

String mensaje = nombre + " >> " + mensajeField.getText();

DatagramPacket paquete = new DatagramPacket(mensaje.getBytes(), mensaje.length(), grupo, PUERTO);

socket.send(paquete);

mensajeField.setText("");

} else if (e.getSource() == salirBtn) {

String mensaje = "\*\*\* " + nombre + " ha salido del chat \*\*\*";

DatagramPacket paquete = new DatagramPacket(mensaje.getBytes(), mensaje.length(), grupo, PUERTO);

socket.send(paquete);

socket.close();

System.exit(0);

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

@Override

public void run() {

byte[] buffer = new byte[1024];

while (true) {

try {

DatagramPacket paquete = new DatagramPacket(buffer, buffer.length);

socket.receive(paquete);

String mensaje = new String(paquete.getData(), 0, paquete.getLength());

chatArea.append(mensaje + "\n");

} catch (Exception e) {

break;

}

}

}

public static void main(String[] args) {

String nombre = JOptionPane.showInputDialog("Introduce tu nombre o nick:");

if (nombre != null && !nombre.trim().isEmpty()) {

ClienteUDP cliente = new ClienteUDP(nombre);

cliente.setVisible(true);

} else {

System.out.println("El nombre no puede estar vacío.");

}

}

}

**EJERCICIO BASICO TCP 3\_7**

**EJERCICIO BASICO UDP 3\_8**