Clase 13 Ejemplo básico de socket monocliente

Cliente:

```
String host ="localhost";
       int puerto = 12345;
       try (Socket socket = new Socket(host,puerto))
       {
           System.out.println("Conectado al servidor");

           BufferedReader entrada = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

           PrintWriter salida = new
PrintWriter(socket.getOutputStream(),true);
salida.println("HOLA SOY EL CLIENTE");
```

Servidor:

```
try (ServerSocket serverSocket = new ServerSocket(puerto))
       {
           System.out.println("Iniciado Servidor en el puerto
"+puerto);

           // esperar a algún cliente
           Socket clienteSocket = serverSocket.accept();
           System.out.println("Cliente conectado al puerto "+puerto);

           BufferedReader entrada = new BufferedReader(new
InputStreamReader(clienteSocket.getInputStream()));

           PrintWriter salida = new
PrintWriter(clienteSocket.getOutputStream(),true);

           String mensajeCliente = entrada.readLine();
```

Clase 14 Empezamos con socket server multicliente

Servidor:

```
try (ServerSocket serverSocket = new ServerSocket(puerto))
       {
```

```
            System.out.println("Servidor levantado");
            while(true)
            {
                Socket clienteSocket = serverSocket.accept();
                System.out.println("Cliente conectado");
                // Lanzar hilo manejador del clienteSocket
                ManejadorSocket clienteManejador = new
ManejadorSocket(clienteSocket);
                new Thread(clienteManejador).start();
```

ManejadorSocket:

```
entrada = new BufferedReader(new
InputStreamReader(socketClient.getInputStream()));
            salida = new
PrintWriter(socketClient.getOutputStream(),true);
            String mensaje;
            while ((mensaje = entrada.readLine()) != null)
            {
                System.out.println("Mensaje recibido del cliente:
"+mensaje);
                salida.println("Soy el manejador del servidor, he
recibido este mensaje:"+mensaje);

            }
```

Clase 15 Fin de socket multicliente identificando

```
import java.net.Socket;
import java.util.UUID;

public class Client2 {
   public static void main(String[] args)
   {
      String host ="localhost";
      int puerto = 8888;

      String idUnico = UUID.randomUUID().toString();
```

Clase 16 Multisockets con semaforo max conexiones

Servidor:

```
try (ServerSocket serverSocket = new ServerSocket(PUERTO))
        {
            System.out.println("Servidor levantado");
            while(true)
            {
```

```
                semaforo.acquire(); // hay hueco?

                Socket clienteSocket = serverSocket.accept();
                System.out.println("Cliente conectado. Conexiones
activas:"+(MAX_CONEXIONES-semaforo.availablePermits()));
                // Lanzar hilo manejador del clienteSocket
                ManejadorSocketAtomic clienteManejador = new
ManejadorSocketAtomic(clienteSocket,semaforo);
                new Thread(clienteManejador).start()
```

Cliente:

```
Scanner entradaConsola = new Scanner(System.in);

            // lectura de la bienvenida del servidor
            byte[] buffer = new byte[1014];
            int bytesLeidos = entrada.read(buffer);
            System.out.println("Servidor:"+new
String(buffer,0,bytesLeidos));

            while (true) {
                String mensaje = entradaConsola.nextLine();
                if (mensaje.equals("salir"))
                {
                    break;
                }
                salida.write(mensaje.getBytes());
                salida.flush();
```

ManejadorSocket:

```
} catch (IOException e) {
            System.out.println("Error en el ManejadorSocket
"+e.getMessage());
        }
        finally
        {
            try {
                clienteSocket.close();
            } catch (Exception e) {
                System.out.println("Error en el ManejadorSocket cerrando
recursos"+e.getMessage());
            }
            semaforo.release();
            System.out.println("cliente desconectado");
        }
```

Clase 17 Multi Sockets simulando un FTP

Cliente:

```
if (!fichero.exists())
            {
                System.out.println("No existe el fichero con ruta
"+rutaFichero);
            }
            else
            {
                String nombreFichero = fichero.getName();
                salida.write(nombreFichero.getBytes());
                salida.flush();

                try (FileInputStream fentrada = new
FileInputStream(fichero))
                {
                    while (((bytesLeidos = fentrada.read(buffer))!= -1))
                    {
                        salida.write(buffer,0,bytesLeidos);
                        salida.flush();
                    }
                }
                System.out.println("Fichero enviado correctamente con la
ruta "+rutaFichero);
            }
```

Manejador:

```
File ficheroDestino = new
File(DIRECTORIO_SERVIDOR+File.separator+nombreFichero);

            try (FileOutputStream fdestino = new
FileOutputStream(ficheroDestino))
            {
                while (((bytesLeidos = entrada.read(buffer))!= -1))
                {
                    fdestino.write(buffer,0,bytesLeidos);
                }
            }
```

Clase 18 Utilización de AtomicInteger con sockets

Servidor:

```
private static final AtomicInteger conexionesActivas = new
AtomicInteger();

try (ServerSocket serverSocket = new ServerSocket(PUERTO))
        {
            System.out.println("Servidor levantado");

            while(true)
            {
                if (conexionesActivas.get()<MAX_CONEXIONES)
                {
                    Socket clienteSocket = serverSocket.accept();
                    int conAct = conexionesActivas.incrementAndGet();
```

Manejador:

```
        } catch (IOException e) {
            System.out.println("Error en el ManejadorSocket
"+e.getMessage());
        }
        finally
        {
            try {
                clienteSocket.close();
            } catch (Exception e) {
                System.out.println("Error en el ManejadorSocket cerrando
recursos"+e.getMessage());
            }
        int conAct= conexionesActivas.decrementAndGet();
        System.out.println("Cliente desconectado. Conexiones activas:
"+conAct);
```

Clase 19 envio obj serializable e inicio de UDP

Serializable:

Clase Entero serializable:

```
public class Entero implements Serializable {
   @Serial
   private static final long serialVersionUID = 1L;
```

```
@Override
   public String toString() {
       return "Entero{" +
               "valor=" + valor +
               ", esPrimo=" + esPrimo +
               '}';
   }
```

Cliente:

```
Entero entero = new Entero(25, false);

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

           salida.writeObject(entero);
```

Servidor:

```
Socket clienteSocket = serverSocket.accept();
               System.out.println("Cliente conectado desde
"+clienteSocket.getInetAddress());

               ObjectInputStream entrada = new
ObjectInputStream(clienteSocket.getInputStream());
               Entero enteroRecibido = (Entero) entrada.readObject();
```

UDP básico:

Cliente:

```
try (DatagramSocket clienSocket = new DatagramSocket())
       {
           InetAddress serverAddress =
InetAddress.getByName("localhost");
           int serverPort = 5000;
           String message = "Hello, Server UDP!";
           byte[] buffer = message.getBytes();
           DatagramPacket packet = new DatagramPacket(buffer,
buffer.length, serverAddress, serverPort);
           clienSocket.send(packet);
```

Servidor:

```
try (DatagramSocket serverSocket = new DatagramSocket(5000)) {
           byte[] buffer = new byte[1024];
           DatagramPacket packet = new DatagramPacket(buffer,
buffer.length);
           System.out.println("Esperando mensaje UDP...");
           serverSocket.receive(packet);
```

```
            String message = new String(packet.getData(), 0,
packet.getLength())
```

Clase 20 Multimensaje con Datagramas UDP

Cliente:

```
while (true)
            {
                String mensaje = scanner.nextLine();
                sendBuffer= mensaje.getBytes();
                DatagramPacket sentPacket = new
DatagramPacket(sendBuffer, sendBuffer.length, serverAddress,5000);
                clientSocket.send(sentPacket);

                if (mensaje.equalsIgnoreCase("exit"))
                {
                    System.out.println("Cerrando cliente......");
                    break;
                }

                DatagramPacket receivePacket = new
DatagramPacket(receiveBuffer, receiveBuffer.length);
                clientSocket.receive(receivePacket);
                String respuestaServidor = new
String(receivePacket.getData(),0,receivePacket.getLength());
                System.out.println("Servidor dice: "+respuestaServidor);

            }
```

Servidor:

```
while(true)
            {
                DatagramPacket packet = new DatagramPacket(buffer,
buffer.length);
                System.out.println("Esperando mensaje UDP...");
                serverSocket.receive(packet);
                String message = new String(packet.getData(), 0,
packet.getLength());
                System.out.println("Mensaje recibido: " + message);

                if(message.equalsIgnoreCase("exit"))
                {
                    System.out.println("El cliente ha terminado la
conexión");
```

```
                    break;
                }

                String response = " Servidor recibió:"+message;
                bufferR = response.getBytes();
                DatagramPacket packetEnvio = new DatagramPacket(bufferR,
bufferR.length,packet.getAddress(),packet.getPort());
                serverSocket.send(packetEnvio);
            }
```

Clase 21 socket multicast

Cliente:

```
String multicastAddress = "239.1.1.1";
        int port = 5000;

        try (MulticastSocket socket = new MulticastSocket(port)) {
            InetAddress group = InetAddress.getByName(multicastAddress);

            NetworkInterface netIf =
NetworkInterface.getByName("wlp2s0");
            socket.joinGroup(new InetSocketAddress(group, port), netIf);
            System.out.println("Cliente unido al grupo multicast.");

            byte[] buffer = new byte[1024];
            DatagramPacket packet = new DatagramPacket(buffer,
buffer.length);

            socket.receive(packet);
            String receivedMessage = new String(packet.getData(), 0,
packet.getLength());
            System.out.println("Mensaje recibido: " + receivedMessage);

            socket.leaveGroup(new InetSocketAddress(group, port),
netIf);
```

Servidor:

```
String multicastAddress = "239.1.1.1";
        int port = 5000;

        try (MulticastSocket socket = new MulticastSocket()) {
            InetAddress group = InetAddress.getByName(multicastAddress);
            String message = "¡ALARMA ESTADO DE SITIO!";
            byte[] buffer = message.getBytes();
```

```
            DatagramPacket packet = new DatagramPacket(buffer,
buffer.length, group, port);
            System.out.println("Enviando mensaje multicast...");
            socket.send(packet);
```