



CHAT MÚLTIPLES CLIENTES - SERVIDOR

MARIO JIMÉNEZ MARSET

ÍNDICE

1. ENUNCIADO - OBJETIVOS.....	3
2. DESARROLLO – PROCEDIMIENTOS.....	3
3. RESULTADOS.....	9

1. ENUNCIADO - OBJETIVOS

En esta práctica se pedía realizar una aplicación de chat multi clientes donde el proceso servidor fuese multihilo, con el objetivo de atender a múltiples clientes.

Tal y como se ha enfocado la práctica, cuando los clientes se conectan, su ip sale en un JComboBox donde se van almacenando todas las de los clientes. Según la ip que se elija, el cliente manda un mensaje. Cuando se escribe la palabra clave “adiós”, este cliente, por mucho que escriba mensajes y presione el botón de “enviar”, no le llegará a ningún otro cliente; tampoco al servidor.

Se ha implementado una interfaz swing, donde cada cliente tiene su ventana propia y el servidor un simple textArea donde visualizar lo que está ocurriendo.

2. DESARROLLO – PROCEDIMIENTOS

Se muestra el código de las dos clases ClienteMultiple y Servidor:

Código clase ClienteMultiple.java:

```
package chatClienteServidor;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;

public class ClienteMultiple {
    public static void main(String[] args) {
        //en el main se llama a la clase MarcoCliente para crear la interfaz del
        cliente,
        //además de hacer que la ventana sea cerrable
        MarcoCliente mimarco=new MarcoCliente();
        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

class MarcoCliente extends JFrame{
    private static final long serialVersionUID = 1L;
    public MarcoCliente(){
        //en el constructor de esta clase que extiende de JFrame, se fijan las
        dimensiones
        //de la ventana, además de llamar a la clase InterfazCliente y ser
        añadida al frame
        //se hace visible y se añade un WindowListener instanciando la clase
        Envio
        setBounds(600,300,280,350);
        InterfazCliente milamina=new InterfazCliente();
        add(milamina);
        setVisible(true);
        addWindowListener(new Envio());
    }
}

class Envio extends WindowAdapter{
    //esta clase extiende de WindowAdapter, añadiendo el método implementado
    windowOpened
    public void windowOpened(WindowEvent e) {
```

```

        //dentro del método se crea el socket (medio de comunicación) con la ip
        local y el número de puerto
        try {
            Socket miSocket=new Socket("192.168.1.18",5000);
            //se instancia un objeto de la clase EnvioDatos
            EnvioDatos datos=new EnvioDatos();
            //se llama al metodo setMensaje
            datos.setMensaje(" online");
            //se crea un ObjectOutputStream, dentro del cual se llama al
            socket con el método getOutputStream()
            ObjectOutputStream paqueteDatos=new
            ObjectOutputStream(miSocket.getOutputStream());
            //se llama al método writeObject() para escribir los datos en el
            ObjectOutputStream
            paqueteDatos.writeObject(datos);
            //se cierra la comunicación cerrando el socket
            miSocket.close();
        }catch(Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}

class InterfazCliente extends JPanel implements Runnable{
    //en esta clase extiende de JPanel ya que los elementos que se creen se van a
    añadir al panel
    //se implementa Runnable ya que se va a crear un Thread y se va a llamar a su
    método start()
    private static final long serialVersionUID = 1L;
    public InterfazCliente(){
        //se crean todos los elementos necesarios y se añaden al panel
        //además, los campos se añaden a los métodos de la clase EnvioDatos
        String nombre=JOptionPane.showInputDialog("Nombre: ");
        JLabel etiquetaNombre=new JLabel("Nombre: ");
        add(etiquetaNombre);
        name=new JLabel();
        name.setText(nombre);
        add(name);
        JLabel texto=new JLabel("Online");
        add(texto);
        ip=new JComboBox<String>();
        add(ip);
        campoChat=new JTextArea(12,20);
        add(campoChat);
        campo1=new JTextField(20);
        add(campo1);
        miboton=new JButton("Enviar");
        //a la clase EnviaTexto se le añade el listener de miboton
        EnviaTexto mievento=new EnviaTexto();
        miboton.addActionListener(mievento);
        add(miboton);
        //se da comienzo al hilo
        Thread mihilo=new Thread(this);
        mihilo.start();
    }
    private class EnviaTexto implements ActionListener{
        //esta clase que implementa ActionListener es la que se llama en la
        anterior clase
        @Override

```

```

public void actionPerformed(ActionEvent e) {
    //al presionar el botón Enviar, se hace todo lo siguiente:
    campoChat.append("\n"+campo1.getText());
    try {
        Socket misocket=new Socket("192.168.1.18",5000);
        //cada mensaje escrito se recoge del campo1 y si es
        //igual a 'adiós', se cierra el socket
        //y no se puede volver a mandar un mensaje
        if(campo1.getText().equals("adios")) {
            misocket.close();
        }
        //se fijan el nombre, ip y mensaje, escribiéndose estos
        //datos en un ObjectOutputStream
        EnvioDatos datos=new EnvioDatos();
        datos.setNombre(name.getText());
        datos.setIp(ip.getSelectedItem().toString());
        datos.setMensaje(campo1.getText());
        ObjectOutputStream paquete_datos=new
        ObjectOutputStream(misocket.getOutputStream());
        paquete_datos.writeObject(datos);
        misocket.close();
    }catch(UnknownHostException e1) {
        System.out.println(e1.getMessage());
    }catch(IOException e2) {
        System.out.println(e2.getMessage());
    }
}

private JTextField campo1;
private JComboBox<String> ip;
private JLabel name;
private JTextArea campoChat;
private JButton miboton;
//al dar comienzo el hilo de la clase InterfazCliente, se ejecuta todo lo siguiente:
@Override
public void run() {
    try {
        @SuppressWarnings("resource")
        //se crea el socket con el número de puerto
        ServerSocket servidor_cliente=new ServerSocket(5000);
        Socket cliente;
        EnvioDatos paqueteRecibido;
        //bucle infinito donde el servidor_cliente espera a que un cliente
        //se conecte
        while(true) {
            cliente=servidor_cliente.accept();
            //cuando el cliente ya está, se lee el flujo de entrada
            //con ObjectInputStream, con readObject()
            ObjectInputStream flujoentrada=new
            ObjectInputStream(cliente.getInputStream());

            paqueteRecibido=(EnvioDatos)flujoentrada.readObject();
            //si el mensaje es el mismo que el que se puso en la
            //clase Envio (que sí es el mismo)
            //se llama al append de campoChat para poner el
            //nombre de quien ha escrito el mensaje y el propio mensaje
            //si no fuese igual, al arraylist de ips se le borrarían
            //todas las ips y se añadiría la del cliente que acaba de conectarse

```

```

        if(!paqueteRecibido.getMensaje().equals(" online")) {

            campoChat.append("\n"+paqueteRecibido.getNombre()+":
"+paqueteRecibido.getMensaje());
        }else {
            ArrayList<String> IpsMenu=new
ArrayList<String>();

            IpsMenu=paqueteRecibido.getIps();
            ip.removeAllItems();
            for(String z:IpsMenu) {
                ip.addIItem(z);
            }
        }
    }catch(Exception e) {
        System.out.println(e.getMessage());
    }
}

//esta clase Serializable declara el arraylist de ips, el nombre, la ip y el mensaje
(variables utilizadas)
//en el resto de clases: es Serializable ya que se utilizan ObjectOutputStream y
ObjectInputStream, que envían
//los datos como bytes, por lo que así se envía todo el paquete de datos de una sola
vez
class EnvioDatos implements Serializable{
    private static final long serialVersionUID = 1L;
    private ArrayList<String> Ips;
    public ArrayList<String> getIps() {
        return Ips;
    }
    public void setIps(ArrayList<String> ips) {
        Ips = ips;
    }
    private String nombre, ip, mensaje;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getIp() {
        return ip;
    }
    public void setIp(String ip) {
        this.ip = ip;
    }
    public String getMensaje() {
        return mensaje;
    }
    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }
}

```

Código clase Servidor.java:

```
package chatClienteServidor;
import javax.swing.*.*;
import java.awt.*.*;
import java.io.*.*;
import java.net.*.*;
import java.util.*.*;

public class Servidor {
    public static void main(String[] args) {
        //en el main se llama a la clase Marco y se fija la ventana como
        cerrable
        Marco marco=new Marco();
        marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
class Marco extends JFrame implements Runnable{
    //esta clase extiende de JFrame para añadir los elementos en el constructor
    //se implementa Runnable ya que se va a crear un Thread y se va a llamar a su
    método start()
    private static final long serialVersionUID = 1L;
    public Marco(){
        //se crean todos los elementos necesarios y se añaden al JFrame
        //se crea un panel donde se añaden los elementos (este panel se
        añade al JFrame)
        //se crea el hilo y comienza su ejecución
        setBounds(1200,300,280,350);
        JPanel milamina= new JPanel();
        milamina.setLayout(new BorderLayout());
        areatexto=new JTextArea();
        milamina.add(areatexto,BorderLayout.CENTER);
        add(milamina);
        setVisible(true);
        Thread mihilo=new Thread(this);
        mihilo.start();
    }
    private JTextArea areatexto;
    @Override
    public void run() {
        //este método sobrescrito va a recibir los mensajes entre los clientes
        try {
            @SuppressWarnings("resource")
            //se crea el socket con el mismo número de puerto que en la
            clase ClienteMultiple
            ServerSocket servidor=new ServerSocket(5000);
            //se crean las variables nombre, ip, mensaje y el arraylist de ips
            String nombre, ip, mensaje;
            ArrayList<String> listaIp=new ArrayList<String>();
            //se crea una instancia de la clase interna EnvioDatos
            EnvioDatos paquete_recibido;
            //bucle infinito
            while(true) {
                //se crea el socket que espera a que el socket servidor
                reciba una petición de cliente
                Socket misocket=servidor.accept();
                //el objeto ObjectInputStream lee los datos del socket
                que ha recibido la información
            }
        }
    }
}
```

```

        ObjectInputStream paquete_datos=new
ObjectInputStream(misocket.getInputStream());
        //se leen con readObject()

        paquete_recibido=(EnvioDatos)paquete_datos.readObject();
        //a las variables antes creadas se les pasa esta
información

        nombre=paquete_recibido.getNombre();
        ip=paquete_recibido.getIp();
        mensaje=paquete_recibido.getMensaje();
        if(!mensaje.equals(" online")) {
            //se añade la información al textArea del
Servidor

            areatexto.append("\n"+nombre+": "+mensaje+"
para "+ip);

            //se vuelve a reenviar la información al
destinatario cliente

            //se cierra la comunicación, el socket
Socket enviaDestinatario=new Socket(ip,5000);
ObjectOutputStream paqueteReenvio=new
ObjectOutputStream(enviaDestinatario.getOutputStream());
            paqueteReenvio.writeObject(paquete_recibido);
            paqueteReenvio.close();
            enviaDestinatario.close();
            misocket.close();
        }else {
            //si el mensaje es igual a online, se recoge la
dirección del socket en una variable
            //esta variable además recoge la ip, que es
remota

            InetAddress
            localizacion=misocket.getInetAddress();
            String
            IpRemota=localizacion.getHostAddress();
            System.out.println("Online "+IpRemota);
            //esta ip se añade a la colección de ips
            listaIp.add(IpRemota);
            //se fijan las ips con el método setIps
            paquete_recibido.setIps(listaIp);
            //este for each tiene como objetivo enviar al
destinatario (cliente) las ips al combobox del cliente
            //y así cargarse dinámicamente
            for(String z:listaIp) {
                System.out.println("Array "+z);
                Socket enviaDestinatario=new

                Socket(z,5000);

                ObjectOutputStream
                paqueteReenvio=new ObjectOutputStream(enviaDestinatario.getOutputStream());

                paqueteReenvio.writeObject(paquete_recibido);
                paqueteReenvio.close();
                enviaDestinatario.close();
                misocket.close();
            }
        }
    } catch (IOException | ClassNotFoundException e) {
        System.out.println(e.getMessage());
    }
}

```

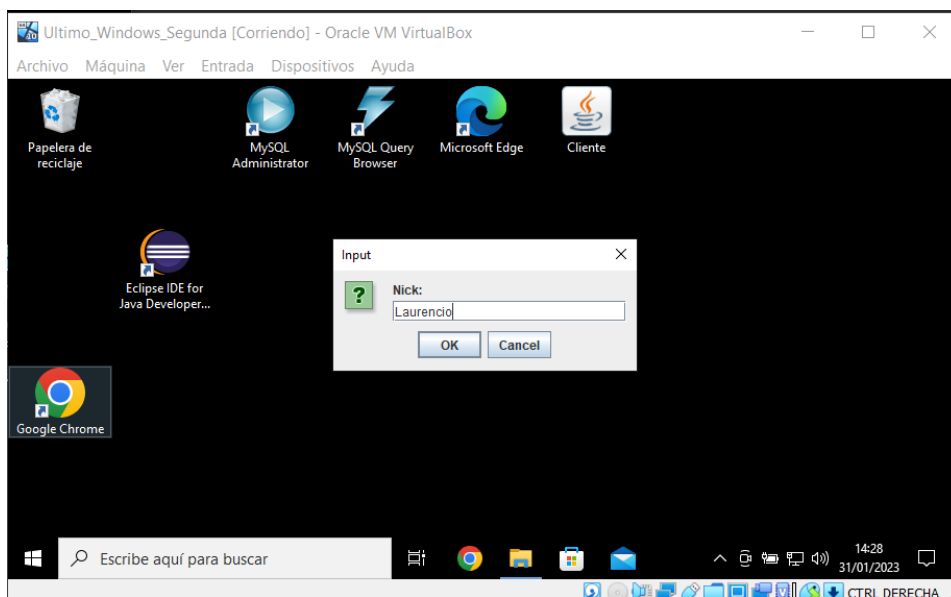
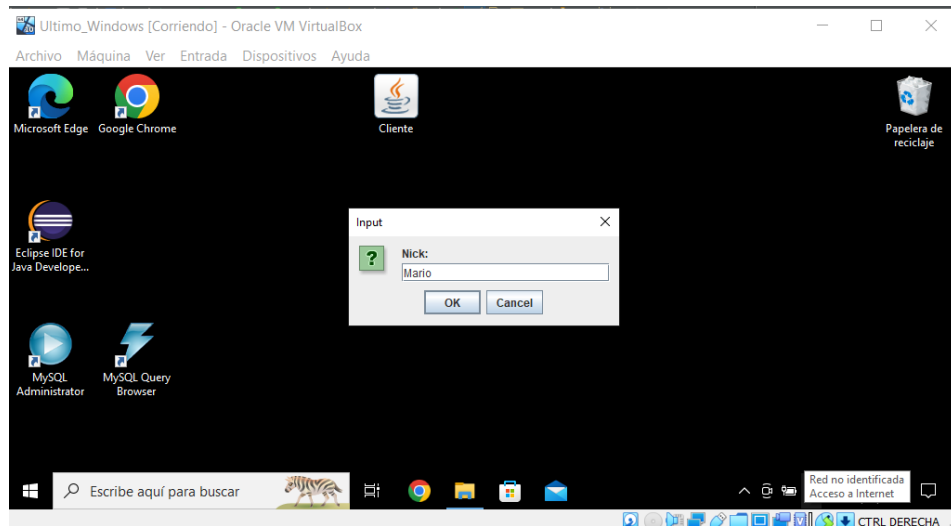


```
}  
}  
}
```

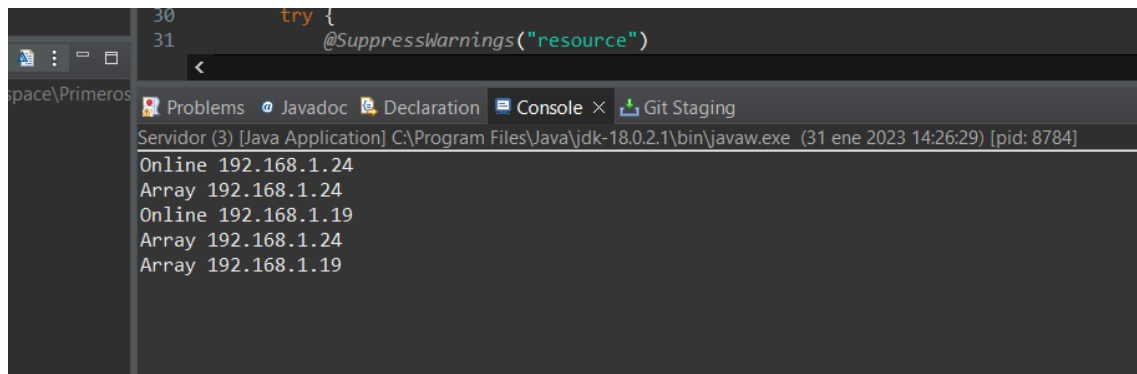
3. RESULTADOS

Finalmente, se muestra un posible resultado de chat entre clientes.

Primero, se introduce el nombre de los clientes en el JOptionPane.

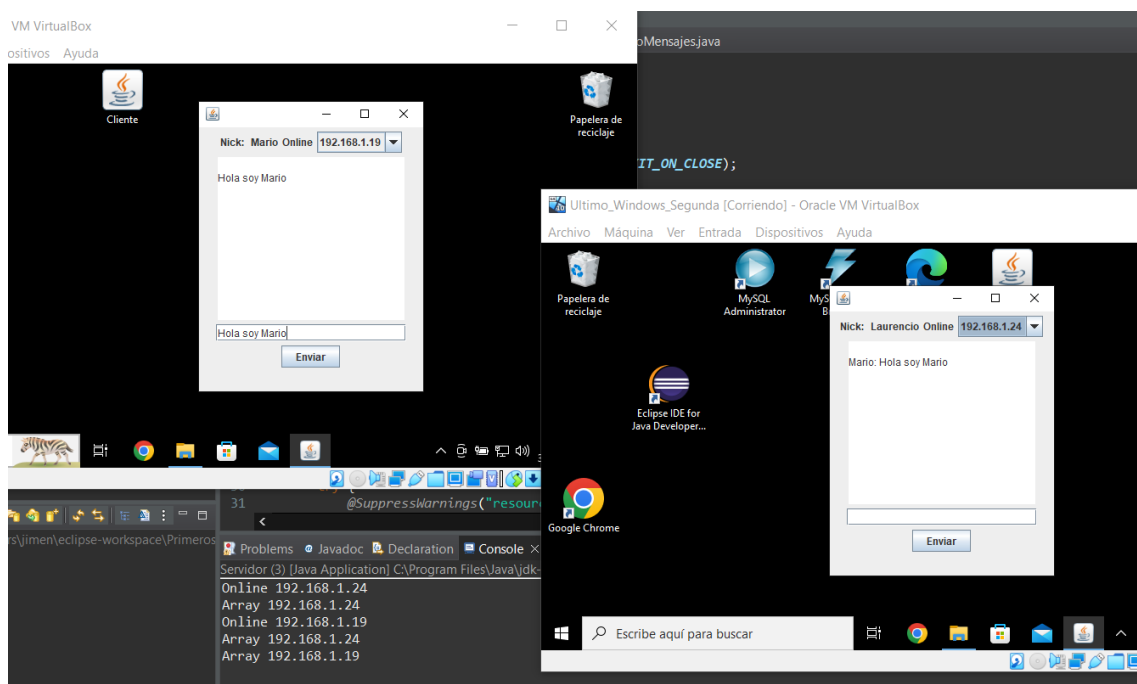


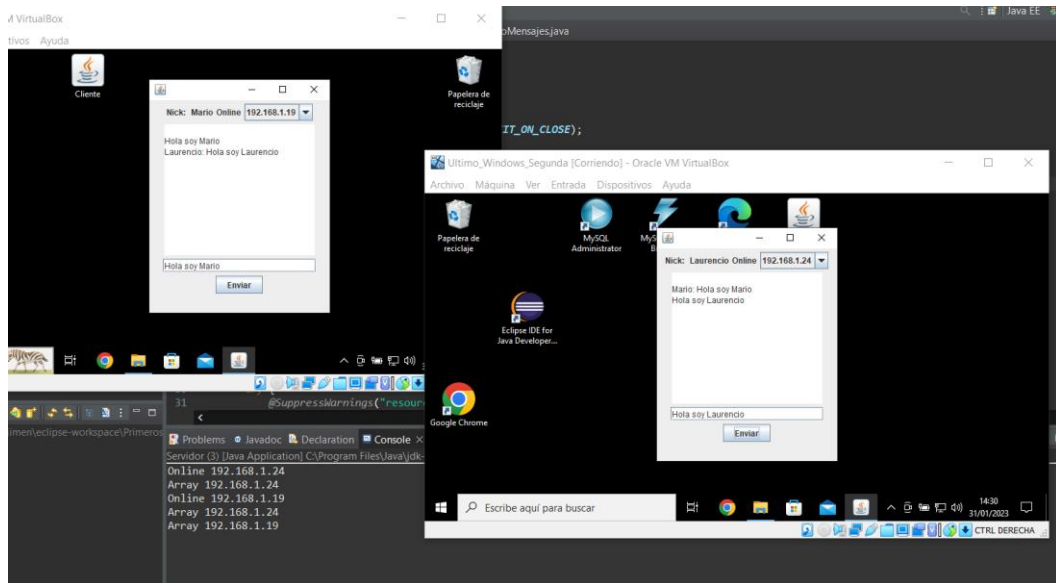
Hecho esto, en la consola del servidor (local, no virtual), se muestran las ips de las máquinas virtuales conectadas (esto se hizo con un arraylist para permitir que muchos clientes se conecten).



```
30      try {
31          @SuppressWarnings("resource")
<
Servidor (3) [Java Application] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.exe (31 ene 2023 14:26:29) [pid: 8784]
Online 192.168.1.24
Array 192.168.1.24
Online 192.168.1.19
Array 192.168.1.24
Array 192.168.1.19
```

Entonces, se abre la ventana donde se pueden enviar mensajes, y el textarea donde se introduce el mensaje que se envía al chat y se muestran también los mensajes de los demás clientes.





Entonces, si un cliente envía la palabra clave 'adiós', se cierra la comunicación, por lo que ya no podrá enviar más mensajes.

