



Centro de Enseñanza Técnica Industrial

Desarrollo de Software

Chat con RMI

Jesús Alberto Aréchiga Carrillo

22310439 5N

Profesor

José Francisco Pérez Reyes

Diciembre 2024

Guadalajara, Jalisco

Introducción

Java RMI (Remote Method Invocation) es una tecnología que permite a las aplicaciones ejecutar métodos en objetos remotos, ubicados en diferentes máquinas dentro de una red. Esto facilita la construcción de aplicaciones distribuidas al proporcionar un mecanismo transparente para la comunicación entre los procesos en distintos dispositivos, manteniendo el paradigma orientado a objetos.

En RMI, el servidor define e implementa una interfaz remota que describe los métodos que pueden ser invocados por los clientes. El servidor registra los objetos remotos en un registro RMI, accesible para los clientes a través de una URL específica. Los clientes buscan estos objetos en el registro y los utilizan como si fueran locales, manejando internamente la comunicación de red mediante serialización de datos y transporte a través de sockets.

Este proyecto aprovecha RMI para crear un sistema de chat distribuido, donde el servidor gestiona las conexiones de los clientes y la transmisión de mensajes. Los clientes interactúan con el servidor para enviar mensajes al chat grupal o iniciar chats privados. RMI simplifica este proceso al ocultar la complejidad de las comunicaciones subyacentes, permitiendo que los desarrolladores se enfoquen en la lógica del negocio en lugar de los detalles técnicos de la red.

En resumen, RMI es una herramienta potente y sencilla para implementar aplicaciones distribuidas en Java, ideal para proyectos como este chat, donde múltiples clientes necesitan interactuar en tiempo real con un servidor centralizado.

Desarrollo

Los códigos para utilizar son:

ServidorChatInterface.java:

```
package Servidor;

import Cliente.ClienteChatInterface;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface ServidorChatInterface extends Remote {
    void registrarCliente(ClienteChatInterface cliente) throws
RemoteException;
    void enviarMensajeBroadcast(String mensaje, String nombreCliente)
throws RemoteException;
    void desregistrarCliente(ClienteChatInterface cliente) throws
RemoteException;
    List<ClienteChatInterface> obtenerClientesConectados() throws
RemoteException;
```

```

    ClienteChatInterface obtenerCliente(String nombreCliente) throws
RemoteException;
}

```

ServidorChat.java:

```

package Servidor;

import Cliente.ClienteChatInterface;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.List;

public class ServidorChat extends UnicastRemoteObject implements
ServidorChatInterface {

    private final List<ClienteChatInterface> clientesConectados;

    public ServidorChat() throws RemoteException {
        super();
        clientesConectados = new ArrayList<>();
    }

    @Override
    public synchronized void registrarCliente(ClienteChatInterface
cliente) throws RemoteException {
        this.clientesConectados.add(cliente);
        System.out.println("Cliente " + cliente.getNombre() + "
registrado.");
    }

    @Override
    public synchronized void enviarMensajeBroadcast(String mensaje,
String nombreCliente) throws RemoteException {
        for (ClienteChatInterface cliente : clientesConectados) {
            if (!cliente.getNombre().equals(nombreCliente)) {
                cliente.recibirMensaje(nombreCliente, mensaje, false); //
false porque no es privado
            }
        }
    }

    @Override

```

```

        public synchronized List<ClienteChatInterface>
obtenerClientesConectados() throws RemoteException {
            return new ArrayList<>(clientesConectados);
        }

        @Override
        public synchronized ClienteChatInterface obtenerCliente(String
nombreCliente) throws RemoteException {
            for (ClienteChatInterface cliente : clientesConectados) {
                if (cliente.getNombre().equals(nombreCliente)) {
                    return cliente;
                }
            }
            return null;
        }

        @Override
        public synchronized void desregistrarCliente(ClienteChatInterface
cliente) throws RemoteException {
            clientesConectados.remove(cliente);
            System.out.println("Cliente " + cliente.getNombre() + "
desregistrado.");
        }

        public static void main(String[] args) {
            try {
                // Crear una instancia del servidor
                ServidorChat servidor = new ServidorChat();

                // Registrar el servidor en el registro RMI
                LocateRegistry.createRegistry(1099);
                Naming.rebind("ServidorChat", servidor);

                System.out.println("Servidor listo y esperando
conexiones...");
            } catch (MalformedURLException | RemoteException e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }
}

```

ClienteChatInterface.java:

```

package Cliente;

import java.rmi.Remote;
import java.rmi.RemoteException;

```

```
public interface ClienteChatInterface extends Remote {  
    void recibirMensaje(String remitente, String mensaje, boolean  
esPrivado) throws RemoteException;  
    String getNombre() throws RemoteException;  
    void setGUI(ClienteChatGUI gui) throws RemoteException;  
}
```

ClienteChat.java:

```
package Cliente;  
  
import java.rmi.RemoteException;  
import java.rmi.server.UnicastRemoteObject;  
  
public class ClienteChat extends UnicastRemoteObject implements  
ClienteChatInterface {  
    private final String nombre;  
    private transient ClienteChatGUI gui;  
  
    public ClienteChat(String nombre) throws RemoteException {  
        super();  
        this.nombre = nombre;  
    }  
  
    // Agregar un método para establecer la referencia de GUI  
    @Override  
    public void setGUI(ClienteChatGUI gui) throws RemoteException {  
        this.gui = gui;  
    }  
  
    @Override  
    public void recibirMensaje(String remitente, String mensaje, boolean  
esPrivado) throws RemoteException {  
        if (gui != null) {  
            gui.mostrarMensaje(remitente, mensaje, esPrivado);  
        } else {  
            System.out.println("GUI es null en ClienteChat. No se puede  
mostrar el mensaje.");  
        }  
    }  
}
```

```

@Override
public String getNombre() throws RemoteException {
    return nombre;
}
}

```

ClienteChatGUI.java:

```

package Cliente;

import Servidor.ServidorChatInterface;
import java.awt.*;
import java.awt.event.*;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.*;
import java.util.List;
import java.util.concurrent.*;
import javax.swing.*;

public class ClienteChatGUI extends JFrame {

    private JTextField campoMensaje;
    private JButton botonEnviar;

    private ClienteChatInterface cliente;
    private ServidorChatInterface servidor;

    private ScheduledExecutorService scheduler;

    private JTabbedPane tabbedPane;
    private Map<String, JTextArea> chats;

    private java.util.List<String> usuariosConectados;

    public ClienteChatGUI(String nombreCliente, String serverIP) {
        super("Chat - " + nombreCliente);

        try {
            // Configurar el cliente y el servidor
            cliente = new ClienteChat(nombreCliente);

            // Conectar al servidor utilizando la IP proporcionada
            String serverURL = "rmi://" + serverIP +
":1099/ServidorChat";

```

```

servidor = (ServidorChatInterface) Naming.lookup(serverURL);
servidor.registrarCliente(cliente);

// Inicializar el mapa de chats
chats = new HashMap<>();

// Crear el JTabbedPane
tabbedPane = new JTabbedPane();

// Agregar el chat grupal a las pestañas
JTextArea areaChatGrupal = new JTextArea();
areaChatGrupal.setEditable(false);
chats.put("Todos", areaChatGrupal);
tabbedPane.addTab("Todos", new JScrollPane(areaChatGrupal));

// Inicializar la lista de usuarios conectados
usuariosConectados = new ArrayList<>();
actualizarListaUsuarios();

// Configurar la interfaz gráfica
campoMensaje = new JTextField();
botonEnviar = new JButton("Enviar");

// Definir el ActionListener
ActionListener enviarMensajeListener = (ActionEvent e) -> {
    enviarMensaje();
};

// Acción al pulsar el botón Enviar
botonEnviar.addActionListener(enviarMensajeListener);

// Acción al presionar Enter en el campo de mensaje
campoMensaje.addActionListener(enviarMensajeListener);

// Disposición de los componentes
JPanel panelInferior = new JPanel(new BorderLayout());
panelInferior.add(campoMensaje, BorderLayout.CENTER);
panelInferior.add(botonEnviar, BorderLayout.EAST);

getContentPane().add(tabbedPane, BorderLayout.CENTER);
getContentPane().add(panelInferior, BorderLayout.SOUTH);

// Añadir la barra de menú
JMenuBar menuBar = new JMenuBar();
JMenu menuChat = new JMenu("Chat");
JMenuItem menuItemNuevoChat = new JMenuItem("Iniciar Chat
Privado");

menuChat.add(menuItemNuevoChat);

```

```

        menuBar.add(menuChat);
        setJMenuBar(menuBar);

        // Acción al seleccionar "Iniciar Chat Privado"
        menuItemNuevoChat.addActionListener(e -> {
            iniciarChatPrivado();
        });

        // Manejar cierre de la ventana
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                cerrarAplicacion();
            }
        });

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 400);
        setVisible(true);

        // Iniciar actualización periódica después de que el
constructor haya terminado
        SwingUtilities.invokeLater(() -> {
            try {
                cliente.setGUI(this);
            } catch (RemoteException e1) {
                System.out.println("Error: " + e1.getMessage());
            }
            iniciarActualizacionPeriodica();
        });

    } catch (NotBoundException | MalformedURLException |
RemoteException e) {
        System.out.println("Error al conectar con el servidor: " +
e.getMessage());
        JOptionPane.showMessageDialog(this, "Error al conectar con el
servidor: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        System.exit(1);
    }
}

private void iniciarActualizacionPeriodica() {
    scheduler = Executors.newSingleThreadScheduledExecutor();
    scheduler.scheduleAtFixedRate(() -> {
        try {
            actualizarListaUsuarios();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }, 0, 1, TimeUnit.SECONDS);
}

```



```

        }, 0, 5, TimeUnit.SECONDS);
    }

    private void actualizarListaUsuarios() {
        try {
            List<ClienteChatInterface> clientes =
servidor.obtenerClientesConectados();
            usuariosConectados = new ArrayList<>();
            for (ClienteChatInterface c : clientes) {
                try {
                    String nombre = c.getNombre();
                    if (!nombre.equals(cliente.getNombre())) {
                        usuariosConectados.add(nombre);
                    }
                } catch (RemoteException e) {
                    System.out.println("Error: " + e.getMessage());
                }
            }
        } catch (RemoteException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private void iniciarChatPrivado() {
        if (usuariosConectados.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No hay usuarios
conectados.", "Información", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        JComboBox<String> comboUsuarios = new
JComboBox<>(usuariosConectados.toArray(String[]::new));
        int opcion = JOptionPane.showConfirmDialog(this, comboUsuarios,
"Iniciar Chat Privado", JOptionPane.OK_CANCEL_OPTION);

        if (opcion == JOptionPane.OK_OPTION) {
            String nombreUsuario = (String)
comboUsuarios.getSelectedItem();
            if (nombreUsuario != null && !nombreUsuario.trim().isEmpty())
{
                try {
                    ClienteChatInterface clienteDestino =
servidor.obtenerCliente(nombreUsuario);
                    if (clienteDestino != null) {
                        // Crear la pestaña si no existe
                        agregarTabChatPrivado(nombreUsuario);
                    } else {
                        JOptionPane.showMessageDialog(this, "El usuario
no está conectado.", "Usuario no encontrado", JOptionPane.ERROR_MESSAGE);

```

```

    }
    } catch (RemoteException ex) {
        System.out.println("Error: " + ex.getMessage());
    }
}

}

}

private void enviarMensaje() {
    try {
        String mensaje = campoMensaje.getText();
        if (!mensaje.isEmpty()) {
            if (mensaje.startsWith("/msj")) {
                // Procesar comando de mensaje privado
                procesarComandoMensajePrivado(mensaje);
            } else {
                // Enviar mensaje al chat activo
                String tabKey =
tabbedPane.getTitleAt(tabbedPane.getSelectedIndex());
                if (tabKey.equals("Todos")) {
                    // Enviar mensaje broadcast
                    servidor.enviarMensajeBroadcast(mensaje,
cliente.getNombre());

                    // Mostrar el mensaje en el chat grupal
                    agregarMensajeATab("Todos", "Yo: " + mensaje);
                } else {
                    // Enviar mensaje directo al chat activo
                    ClienteChatInterface clienteDestino =
servidor.obtenerCliente(tabKey);
                    if (clienteDestino != null) {
                        clienteDestino.recibirMensaje(cliente.getNomb
re(), mensaje, true);

                        // Mostrar el mensaje en la pestaña
correspondiente

                        agregarMensajeATab(tabKey, "Yo (privado): " +
mensaje);
                    } else {
                        JOptionPane.showMessageDialog(this, "Usuario
no encontrado.", "Error", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        }
        campoMensaje.setText("");
    }
} catch (RemoteException e) {
    System.out.println("Error: " + e.getMessage());
}
}

```

```

private void procesarComandoMensajePrivado(String comando) {
    // Eliminar el prefijo "/msj" y cualquier espacio al inicio
    String contenido = comando.substring(4).trim();
    int primerEspacio = contenido.indexOf(' ');
    if (primerEspacio == -1) {
        JOptionPane.showMessageDialog(this, "Formato incorrecto. Uso: /msj usuario mensaje", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    String nombreUsuario = contenido.substring(0,
primerEspacio).trim();
    String mensaje = contenido.substring(primerEspacio).trim();
    if (nombreUsuario.isEmpty() || mensaje.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Formato incorrecto. Uso: /msj usuario mensaje", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    try {
        if (!nombreUsuario.equals(cliente.getNombre())) {
            ClienteChatInterface clienteDestino =
servidor.obtenerCliente(nombreUsuario);
            if (clienteDestino != null) {
                // Enviar el mensaje privado
                clienteDestino.recibirMensaje(cliente.getNombre(),
mensaje, true);

                // Mostrar el mensaje en la pestaña correspondiente
                agregarMensajeATab(nombreUsuario, "Yo (privado): " +
mensaje);
            } else {
                JOptionPane.showMessageDialog(this, "El usuario no
está conectado.", "Usuario no encontrado", JOptionPane.ERROR_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(this, "No puedes enviarte
mensajes a ti mismo.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    } catch (RemoteException e) {
        System.out.println("Error: " + e.getMessage());
    }
}

public void mostrarMensaje(String remitente, String mensaje, boolean
esPrivado) {
    SwingUtilities.invokeLater(() -> {
        String tabKey = esPrivado ? remitente : "Todos";
        agregarMensajeATab(tabKey, remitente + (esPrivado ? "
(privado)" : "") + ": " + mensaje);
    });
}

```

```

private void agregarTabChatPrivado(String nombreUsuario) {
    if (!chats.containsKey(nombreUsuario)) {
        JTextArea areaChat = new JTextArea();
        areaChat.setEditable(false);
        chats.put(nombreUsuario, areaChat);
        tabbedPane.addTab(nombreUsuario, new JScrollPane(areaChat));
    }
}

private void agregarMensajeATab(String tabKey, String mensaje) {
    agregarTabChatPrivado(tabKey);
    JTextArea areaChat = chats.get(tabKey);
    areaChat.append(mensaje + "\n");
}

private void cerrarAplicacion() {
    if (scheduler != null && !scheduler.isShutdown()) {
        scheduler.shutdown();
    }
    try {
        servidor.desregistrarCliente(cliente);
    } catch (RemoteException e) {
        System.out.println("Error: " + e.getMessage());
    }
    System.exit(0);
}

public static void main(String[] args) {
    // Solicitar la dirección IP
    String serverIP = JOptionPane.showInputDialog(null, "Ingrese la
dirección IP del servidor:", "Conexión al Servidor",
JOptionPane.QUESTION_MESSAGE);
    if (serverIP == null || serverIP.trim().isEmpty()) {
        System.out.println("La dirección IP no puede estar vacía.");
        System.exit(0);
    }

    // Solicitar el nombre del cliente
    String nombreCliente = JOptionPane.showInputDialog("Ingrese su
nombre:");
    if (nombreCliente != null && !nombreCliente.isEmpty()) {
        new ClienteChatGUI(nombreCliente, serverIP);
    } else {
        System.out.println("El nombre no puede estar vacío.");
        System.exit(0);
    }
}
}

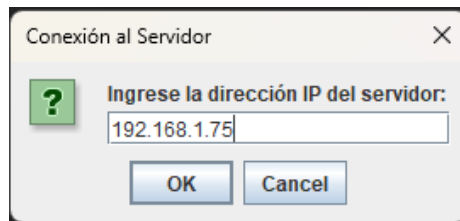
```

Se ejecuta el código de ServidorChat para empezar el servidor y que esté listo para recibir los mensajes.

Servidor listo y esperando conexiones...

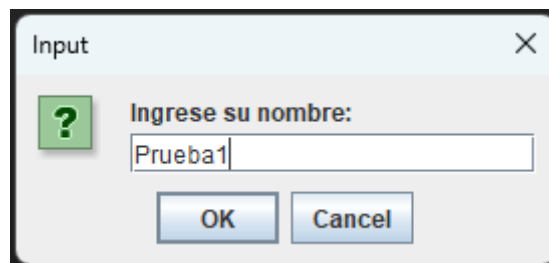
En este punto ya se pueden conectar los clientes para mandar los mensajes.

Al ejecutar el cliente, lo primero es ingresar la dirección IP que en este caso es 192.168.1.75.



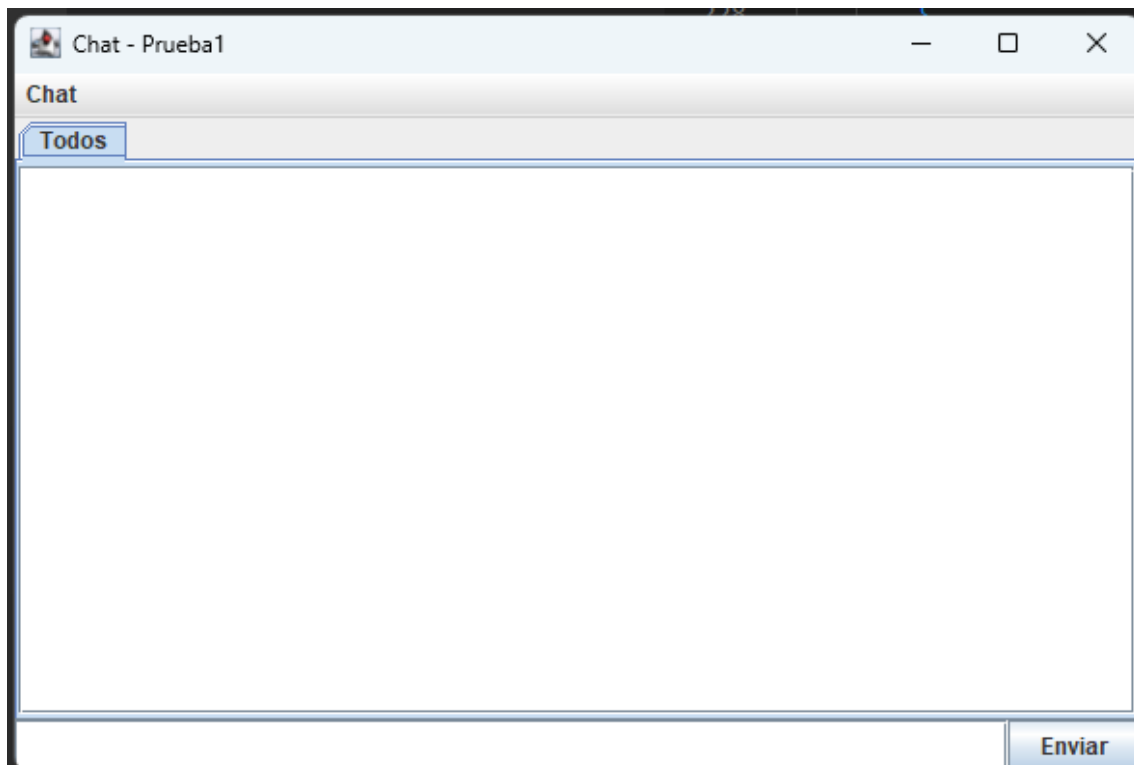
A dialog box titled "Conexión al Servidor" with a close button (X) in the top right corner. It contains a green square icon with a white question mark. To the right of the icon is the text "Ingrese la dirección IP del servidor:". Below this text is a text input field containing "192.168.1.75". At the bottom of the dialog are two buttons: "OK" and "Cancel".

Después es ingresar el nombre con el que se va a identificar.



A dialog box titled "Input" with a close button (X) in the top right corner. It contains a green square icon with a white question mark. To the right of the icon is the text "Ingrese su nombre:". Below this text is a text input field containing "Prueba1". At the bottom of the dialog are two buttons: "OK" and "Cancel".

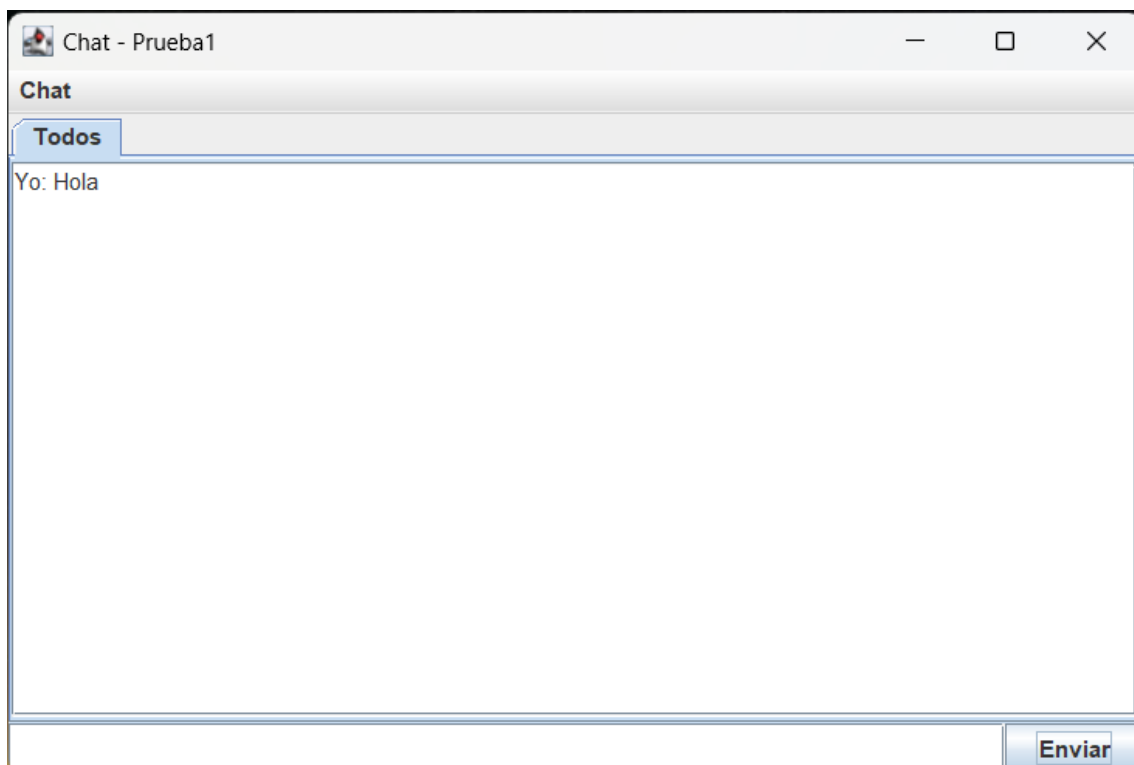
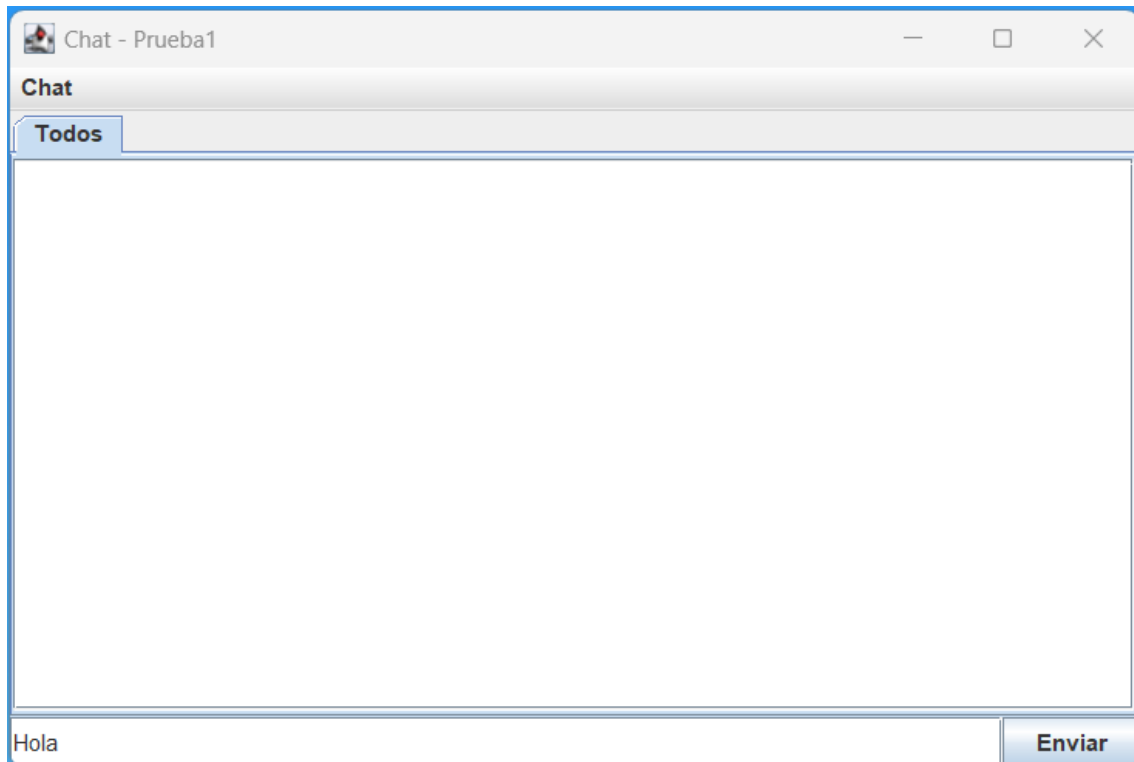
Después se abre la interfaz para mandar mensajes.

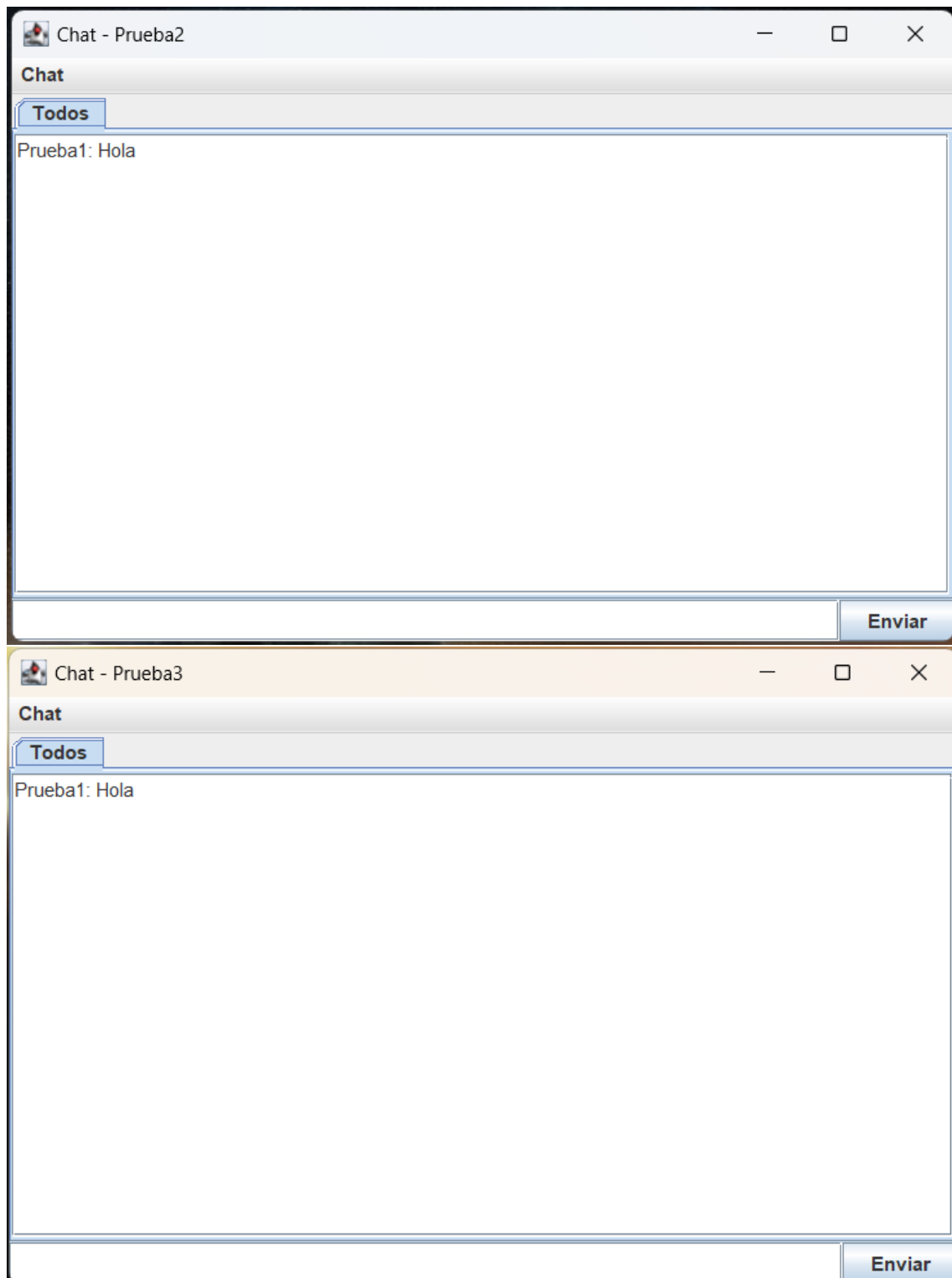


A window titled "Chat - Prueba1" with standard window controls (minimize, maximize, close) in the top right corner. The window has a tab labeled "Chat" and a sub-tab labeled "Todos". Below the tabs is a large, empty text area for messages. At the bottom of the window is a text input field and a button labeled "Enviar".

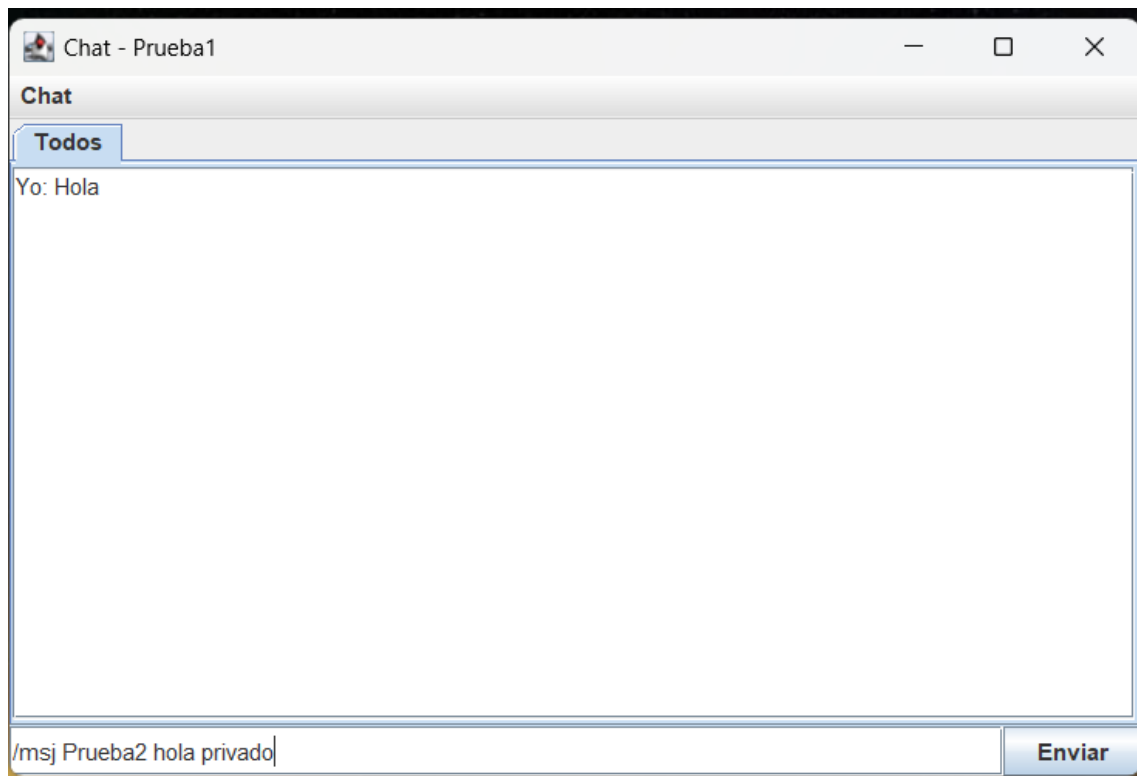
En este punto ya es posible empezar a enviar mensajes, pero es necesario que haya más clientes conectados, por lo que es necesario repetir el proceso hasta que se tenga el número de clientes conectados necesarios.

Una vez que se tengan todos los clientes conectados, se puede empezar a enviar mensajes ya sea para todos o para algún usuario en específico.

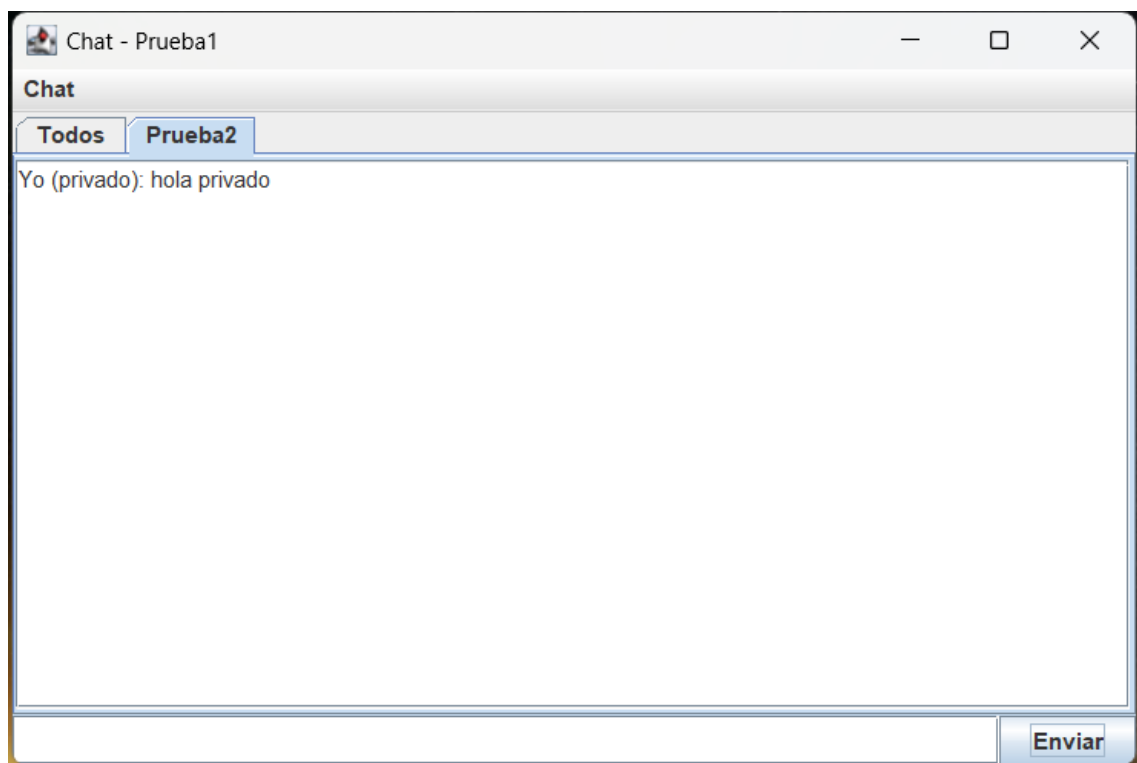




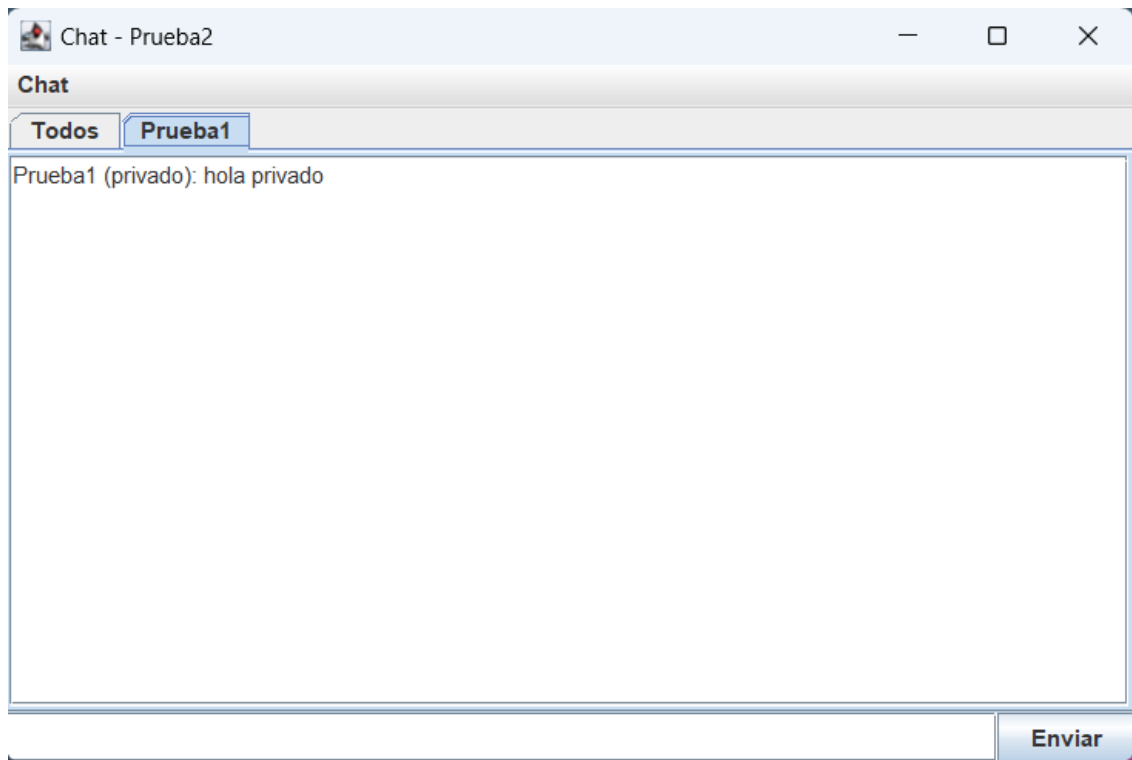
Para enviar un mensaje en privado hay 2 maneras. La primera es con el comando “/msj usuario mensaje” donde “usuario” es el nombre del usuario al que se le va a enviar el mensaje y “mensaje” es el texto que se le va a enviar.



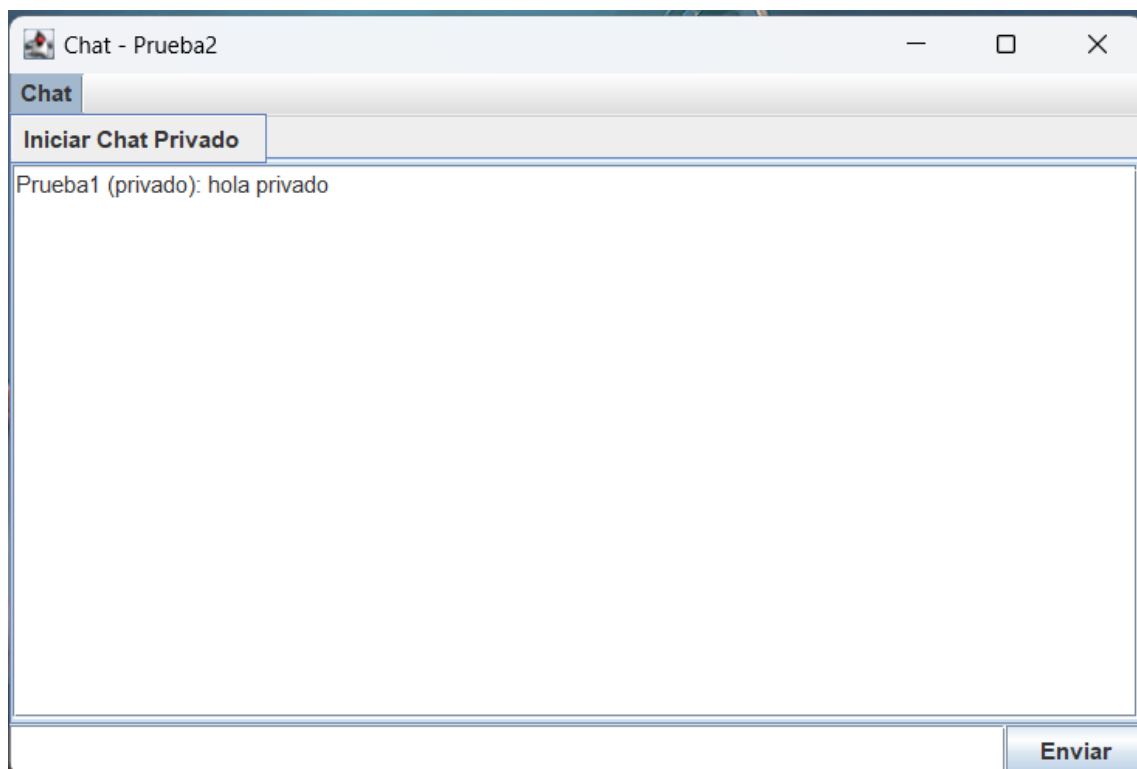
El comando ya está preparado para que se envíe el mensaje privado.



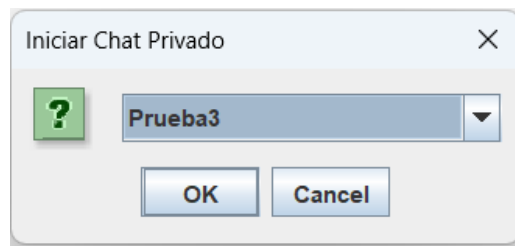
Ya se abrió el chat con el usuario Prueba2, en el chat de Prueba2.



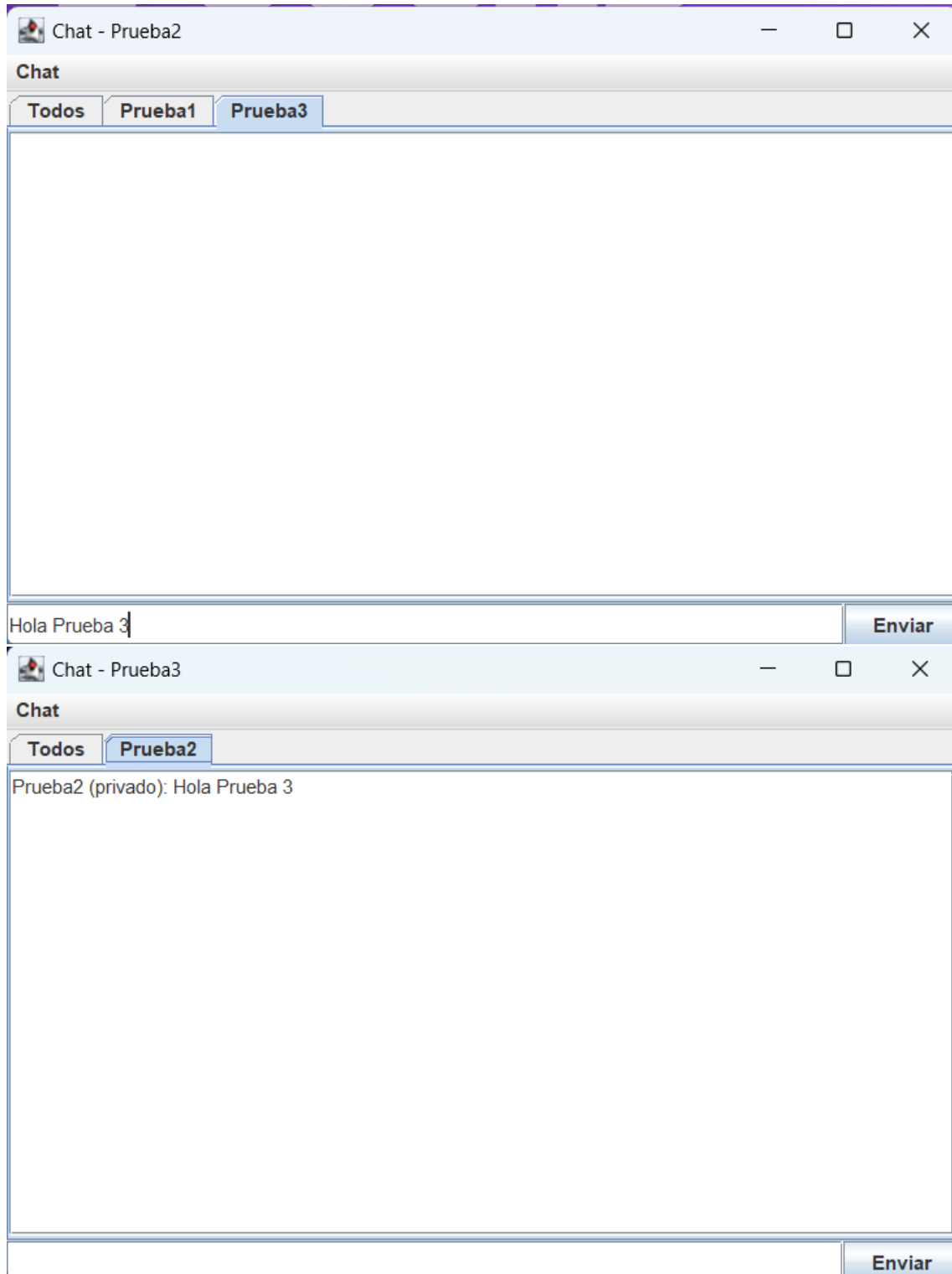
La otra manera es en el menú de chat en la parte de arriba, al darle clic, se despliega un menú con la opción para enviar un mensaje en privado.



Después de darle clic a “Iniciar Chat Privado”, se puede seleccionar un usuario para mandar un mensaje privado.



En este caso el usuario de Prueba2 le va a enviar un mensaje privado a Prueba3.



Conclusiones:

Java RMI es una herramienta poderosa para construir sistemas distribuidos que necesitan comunicar procesos en diferentes máquinas. Al abstraer los detalles de la red, RMI permite a los desarrolladores trabajar con objetos remotos de manera similar a los locales, facilitando la implementación de arquitecturas cliente-servidor complejas.

La seguridad y la gestión de excepciones son aspectos críticos en RMI, ya que la comunicación distribuida puede estar sujeta a problemas como fallos en la red o ataques externos. A pesar de estas consideraciones, RMI sigue siendo una opción sólida para sistemas basados en Java, aunque su popularidad ha disminuido frente a tecnologías modernas como gRPC y REST. Sin embargo, para proyectos Java específicos o legados, RMI proporciona una solución madura y eficiente.