*****Instituto Politécnico Nacional***

***Escuela Superior de Cómputo***

*Aplicaciones para Comunicaciones de Red*

***Practica 4: Servicio de Chat***

***Integrantes:***

* *Ornelas García Luis Ángel*
* *Sampayo Hernández Mauro*

***Grupo:*** *3CM5*

***Profesor:*** *Moreno Cervantes Axel Ernesto*

**Introducción:**

Hoy día, en muchos tipos de aplicaciones se cuenta con un servicio de chat, ya sea con fines de diversión o de negocio. Este tipo de servicio es muy socorrido cuando se trata de brindar una comunicación más personalizada que la brindada por medios tales como correo electrónico, o foros. En el chat la comunicación puede ser fluida, es decir, en tiempo real y además pueden interactuar dos o más personas a la vez.

**Desarrollo:**

* **Mensaje**

Este paquete contiene la clase Mensaje la cual definirá cada uno de los mensajes que sean enviados por cada usuario en el chat.

Este objeto tiene 4 atributos (tipo, usuarioOrigen, usuarioDestino y mensaje) y 3 maneras de ser inicializado que dependerán del tipo de mensaje que esté siendo enviado.

Para los mensajes de inicio y fin de sesión, se inicializará el objeto con solo dos atributos: tipo y usuarioOrigen. En “tipo” se pondrá el texto “INICIO” o “FIN” dependiendo de si el mensaje indica que un usuario ha iniciado o finalizado sesión; y en “usuarioOrigen” se pondrá el nombre del usuario en cuestión.

Para los mensajes grupales, se inicializará el objeto con tres atributos: tipo, usuarioOrigen y mensaje. “Tipo” en este caso siempre será definido con el texto “PUBLICO”, “usuarioOrigen” tendrá el nombre del cliente que mandó el mensaje, y “mensaje” contendrá el mensaje enviado en cuestión.

Finalmente, para los mensajes privados, se inicializará el objeto con cuatro atributos: tipo, usuarioOrigen, usuarioDestino y mensaje. “Tipo” siempre será definido con el texto “PRIVADO”, “usuarioOrigen” tendrá el nombre del cliente que mandó el mensaje, “usuarioDestino” tendrá el nombre del usuario al cual se le envía el mensaje, y “mensaje” contendrá el mensaje en cuestión.

* **ServidorMulticast**

Dentro de este paquete se lleva a cabo la lógica del Servidor, el cual llevará a cabo la gestión de usuarios que se hallen conectados, actualizándose al momento cada que un usuario inicie o finalice sesión. Este paquete solo tiene una clase definida la cual es **“Servidor”**.

Esta clase tiene definidas como variables globales un socket multicast y una serie de Strings estáticos que definen el puerto y la dirección a la que se conectará el socket (en este caso será el puerto 4000 y la dirección 230.1.1.1), un String estático que define el tamaño del buffer que será usado para enviar y recibir objetos desde un socket multicast, y un ArrayList en donde se guardaran los nombres de los usuarios que se encuentren conectados, y que se actualizará cada que un usuario se incorporé o abandone el chat.

Al momento de iniciar la ejecución del servidor se realiza una resolución de la dirección para determinar el grupo al que se conectarán los clientes y el servidor y se inicializará el puerto, la interfaz de red y la dirección de conexión. Después se inicializará un ciclo infinito, en el cual el servidor estará recibiendo constantemente todos los objetos de tipo Mensaje que sean mandados por todos los clientes, para analizar posteriormente el tipo de mensaje que son.

En caso de que el mensaje sea de tipo “INICIO”, se añadirá a la lista el nombre del usuario que ha iniciado sesión, y se enviará de vuelta a todos los clientes conectados tanto el número de usuarios conectados, así como también cada uno de los elementos de la lista de usuarios del servidor. En caso de que el mensaje sea de tipo “FIN”, se realizará lo mismo que cuando el mensaje es de tipo “INICIO” con la única diferencia de que el nombre del usuario que a finalizado sesión será removido de la lista.

Para cuando se reciban mensajes de tipo “PUBLICO” y “PRIVADO”, el servidor no realizará ninguna acción, y pasará a esperar a recibir el mensaje siguiente.

* **ClienteMulticast**

Dentro de este paquete están todas las clases que lleva a cabo la lógica del inicio y fin de sesión de cada uno de los usuarios, el envío y recibimiento de mensajes públicos y privados de un usuario a otro, y el entorno gráfico del chat donde se muestran los mensajes con la inclusión de emoticonos, los usuarios conectados y un espacio en el cual el usuario podrá escribir un mensaje para posteriormente enviarlo.

**Clase ClienteMulticast:**

La clase “ClienteMulticast” se encarga de crear e inicializar tanto el componente *Cliente*, como también el hilo que ejecutará el método **run()** implementado dentro del componente “Cliente” cada que este sea inicializado.

**Componente Login:**

El componente “Login” es un JDialog cuya ventana padre es la del componente *Cliente*, el cual le pedirá al cliente que escriba un nombre de usuario, el cuál será validado al momento en que el botón “Login” sea presionado, y en caso de ser aceptado esté será guardado en una variable String universal llamada “nombreUsuario”.

En caso de que el nombre ingresado por el cliente no sea válido, el programa mandará un mensaje de error y terminará su ejecución. Los dos únicos casos que pueden ser considerados un nombre de usuario no válido son que el campo sea dejado en blanco, o que se deje el texto por defecto definido en el JTextField el cual es “Nombre de Usuario.

Este componente también cuenta con 2 métodos llamados **“initComponents()”** y **“Componentes()”** que se encargan de inicializar y definir los elementos gráficos del JDialog.

**Componente Cliente:**

El componente “Cliente” es un JFrame que implementa el método Runnable, y que se encarga de llevar a cabo toda la lógica detrás de las operaciones y elementos gráficos que se mostrarán en el chat.

Este componente tiene definido como variables globales, una serie de emoticonos en formato ASCII declaradas dentro de cadenas de texto(Strings) estáticas, tanto un socket multicast como una serie de Strings estáticos que definen el puerto y la dirección a la que se conectará el socket (en este caso el puerto 4000 y la dirección 230.1.1.1), un String estático que define el tamaño del buffer que será usado para enviar y recibir objetos desde un socket multicast, un String donde se guardará el nombre de usuario del cliente una vez sea establecido, un ArrayList que contendrá el nombre de todos los usuarios que se encuentren conectados en cada momento, un objeto de tipo Mensaje, y dos Strings que funcionan como auxiliares para la creación de mensajes de tipo Privado, y los cuales guardarán el tipo de mensaje que será creado (si es privado el String guardará la cadena de texto “Privado”, en caso contrario será null) y el usuario destinatario al que le será enviado el mensaje privado en caso de enviarse uno.

Al inicializar un hilo de este componente, se realizará una resolución de la dirección a la que conectaremos nuestro grupo de clientes, y al que estará conectado el servidor. Tras esto se ejecutan los métodos **LogIn()** e i**nitComponentes()** los cuales se encargan de realizar la conexión del socket del cliente dentro de la dirección y puertos especificados, y de la declaración e inicialización de los elementos gráficos del chat respectivamente.

También se declaró un oyente de Ventana, el cual inicializa una Ventana de Confirmación cada que el usuario presione el botón para cerrar la ventana, y que le preguntará al usuario si realmente desea salir del chat. Si el usuario selecciona la opción “Si”, se ejecutará el método **“LogOut()”** y después cerrará y finalizará la el programa. Por el contrario, si el usuario selecciona la opción de “No”, la Ventana de Confirmación será cerrada y el programa seguirá corriendo normalmente.

En el método **run()** se estará ejecutando infinitamente la función **“Recibir Mensaje()”.**

* **Método LogIn()**

Al ejecutarse este método, se inicializa el componente *Login*, del cual se accederá a su atributo “nombreUsuario” siempre y cuando este ya haya sido validado y aceptado en el componente *Login*, y se guardará dentro de la variable definida para el nombre del usuario del componente *Cliente*.

Después, se inicializa un objeto *Mensaje* de tipo “INICIO” (el cual indica que un usuario inició sesión) y un socket multicast por medio del cual se mandará el mensaje de tipo “INICIO” para así indicarles a todos los usuarios conectados y al servidor que un nuevo usuario ha iniciado sesión.

* **Método LogOut()**

Cuando se accede a este método, se inicializa un objeto *Mensaje* de tipo “FIN” (que indica que un usuario finalizó sesión) y un socket multicast por medio del cual se mandará el mensaje de tipo “FIN” para así indicarles a todos los usuarios conectados y al servidor que un nuevo usuario ha finalizado sesión.

**Método RecibirMensaje()**

Este método espera a recibir un objeto de tipo *Mensaje*, para después determinar qué tipo de mensaje que son y así realizar cierta acción en el chat.

Si el mensaje recibido es de tipo “INICIO”, se inicializarán nuevos paquetes de datagrama por medio de los cuales se recibirá el número de usuarios que hay en la ArrayList de clientes del servidor, así como también los elementos de esta misma. Estos elementos serán incorporados a la ArrayList de usuarios del cliente, siempre y cuando estos no se hallen previamente en dicha lista.

Posterior a esto, se ejecutará el método **“ActualizarUsuarios()”** y en el chat de los usuarios que se hallen conectados en ese momento, se mostrará una cadena de texto indicando el inicio del sesión de un nuevo usuario.

Si el mensaje es de tipo “PUBLICO”, se llamará al método **“Emoticonos()”**, pasándole el mensaje como parámetro, y la cadena de texto devuelta por dicho método (el mismo mensaje pero con la inclusión de emoticones) se mostrará en el chat de todos los usuarios que se encuentren conectados.

Si el mensaje es de tipo “PRIVADO”, se realizará el mismo procedimiento que si este fuera de tipo “PUBLICO”, con la diferencia de que este proceso solo será realizado por el usuario que haya enviado el mensaje y el usuario al cual le haya sido enviado dicho mensaje. Dicho mensaje solo se mostrará en el chat de esos dos usuarios.

Finalmente, si el mensaje es de tipo “FIN”, se realizará el mismo procedimiento que si este fuera de tipo “INICIO”, con la diferencia de que la lista de usuarios del cliente, será vaciada en su totalidad, para luego guardar todos los elementos de la lista de usuarios enviada desde el servidor.

* **Método ActualizarUsuarios()**

Este método actualizará la lista de usuarios conectados cada que un usuario inicie o finalice sesión.

Al ser ejecutado este método, todos los elementos que haya en el JPanel llamado “Usuarios” (en donde se muestra una lista con los usuarios conectados y que es definida en el método **“initComponents()”**) serán removidos, para después declarar un JTextField no editable, que contendrá el nombre de usuario que el Cliente estableció previamente.

Tras esto se declararán diversos JTexrField no editables, los cuales contendrán el nombre de todos los usuarios contenidos en el ArrayList de usuarios conectados, los cualés tendrán asociados un menú desplegable que tendrá una única opción llamada “Mensaje Privado”, la cual establecerá a la variable global “tipo” como “PRIVADO” y la variable lobal “usuarioDestino” con el nombre de usuario asociado al JTextFiel del cual se haya desplegado el menú.

Finalmente se añadirán al JPanel “Usuarios” todos los JTextFields que hayan sido creados.

* **Método Emoticonos()**

Este método recibe el mensaje que haya sido recibido por el Cliente como cadena de texto, la cual será analizada para determinar si contiene subcadenas que puedan representar un emoticono, y en caso de tenerlas, cambiarlas por el emoticono correspondiente.

Los emiticonos y sus respectivas subcadenas que los representan se enlistan a continuación:

* “owo” representa al emoticono “oωo”
* “->” representa al emoticono “→”
* “<->” representa al emoticono “↔”
* “<-” representa al emoticono “←”
* “:)” representa al emoticono “☺”
* “:/” representa al emoticono “😐”
* “:(” representa al emoticono “☹”
* “emt-:plane” que representa al emoticono “✈”
* “emt-:star” que representa al emoticono “✪”
* “emt-:commie” que representa al emoticono “☭”

El análisis del mensaje consta en tomar cada una de las letras de la cadena de texto para compararlas con las subcadenas que representan a los emoticonos y comprobar si coinciden con alguna de ellas siempre y cuando estén separadas del resto de palabras por espacios (“ “).

Cada una de las letras que hayan sido analizadas, serán guardadas en una nueva cadena de texto, y en caso de que un grupo de letras represente un emoticono, estas serán reemplazadas por el emoticono correspondiente dentro de la nueva cadena de texto.

Una vez finalizado el análisis del mensaje, se devolverá el mensaje con los emoticonos ya añadidos.

* **Metodo initComponents()**

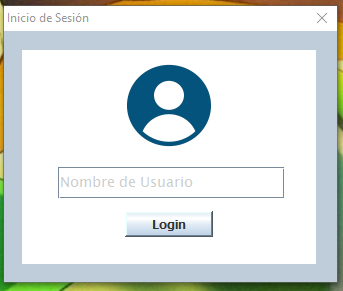
Este método define todos los elementos gráficos del chat, los cuales son un JTextPane en donde se mostrarán todos los mensajes del chat, un JTextArea en donde el usuario puede escribir un mensaje para después enviarlo, un JPanel llamado “Usuarios” en donde se enlistan todos los usuarios se encuentren conectados, y dos JLabels, los cuales brindarán información de la interfaz de usuario y el tipo de mensaje que estemos enviando en ese momento.

También se cuenta con un botón llamado **“Enviar”**, que al ser presionado tomará el texto que haya sido escrito dentro del JTextArea (siempre y cuando no esté vacía), y lo inicializará dentro de un objeto tipo *Mensaje* junto a las configuraciones correspondientes de acuerdo al tipo de mensaje que sea y posteriormente será enviado por medio de un socket multicast a los demás usuarios.

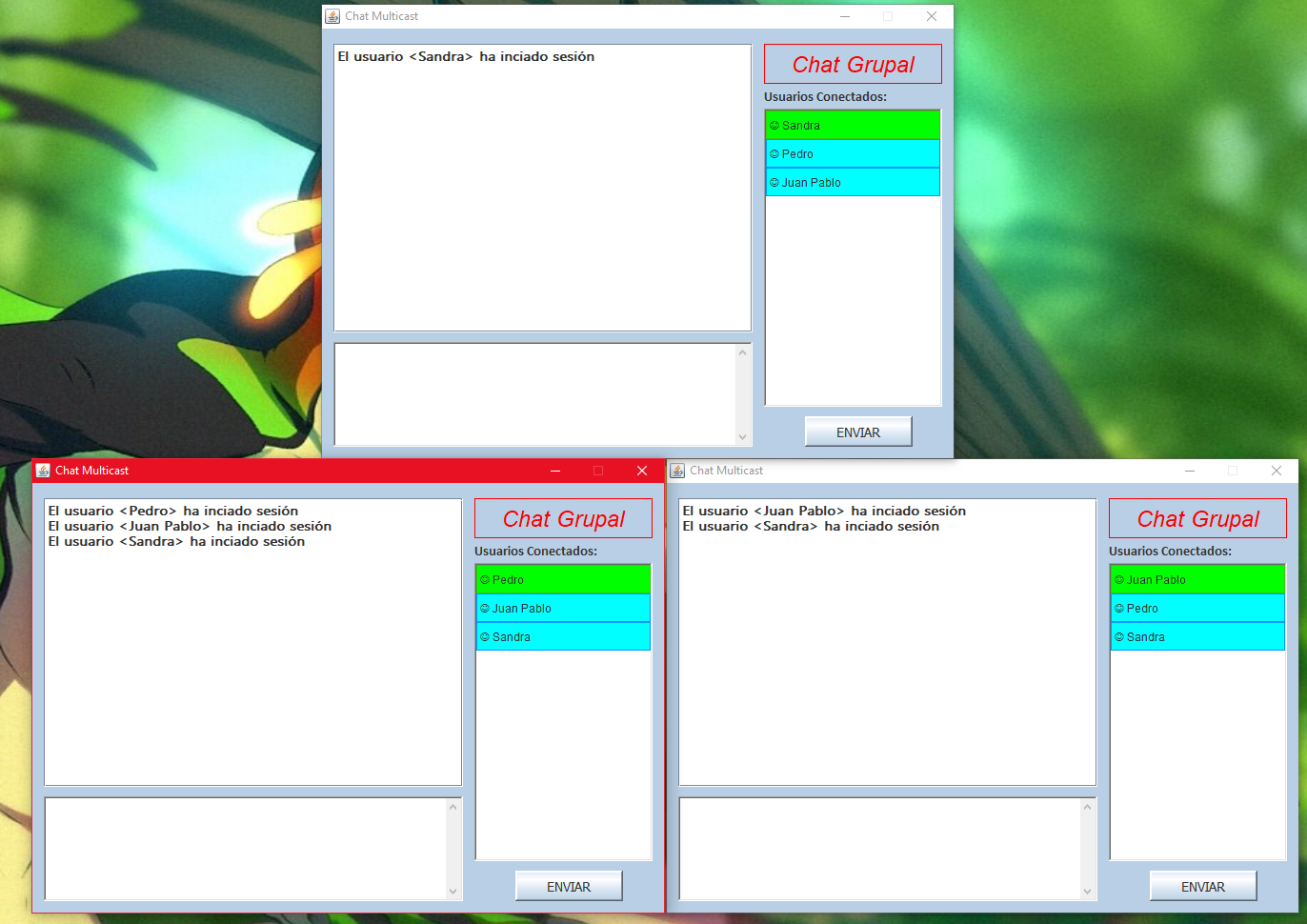
El JTextPane donde se mostrarán los mensajes del chat, tiene asociado un Menú Desplegable el cual tendrá una única opción, la cual es **“Mensaje al Grupo”**, y que, al ser seleccionada, hará que los siguientes mensajes que sean enviados por el usuario sean de tipo “PUBLICO”.

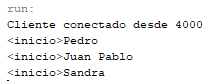
**Pruebas:**

* **Pantalla de inicio de sesión**

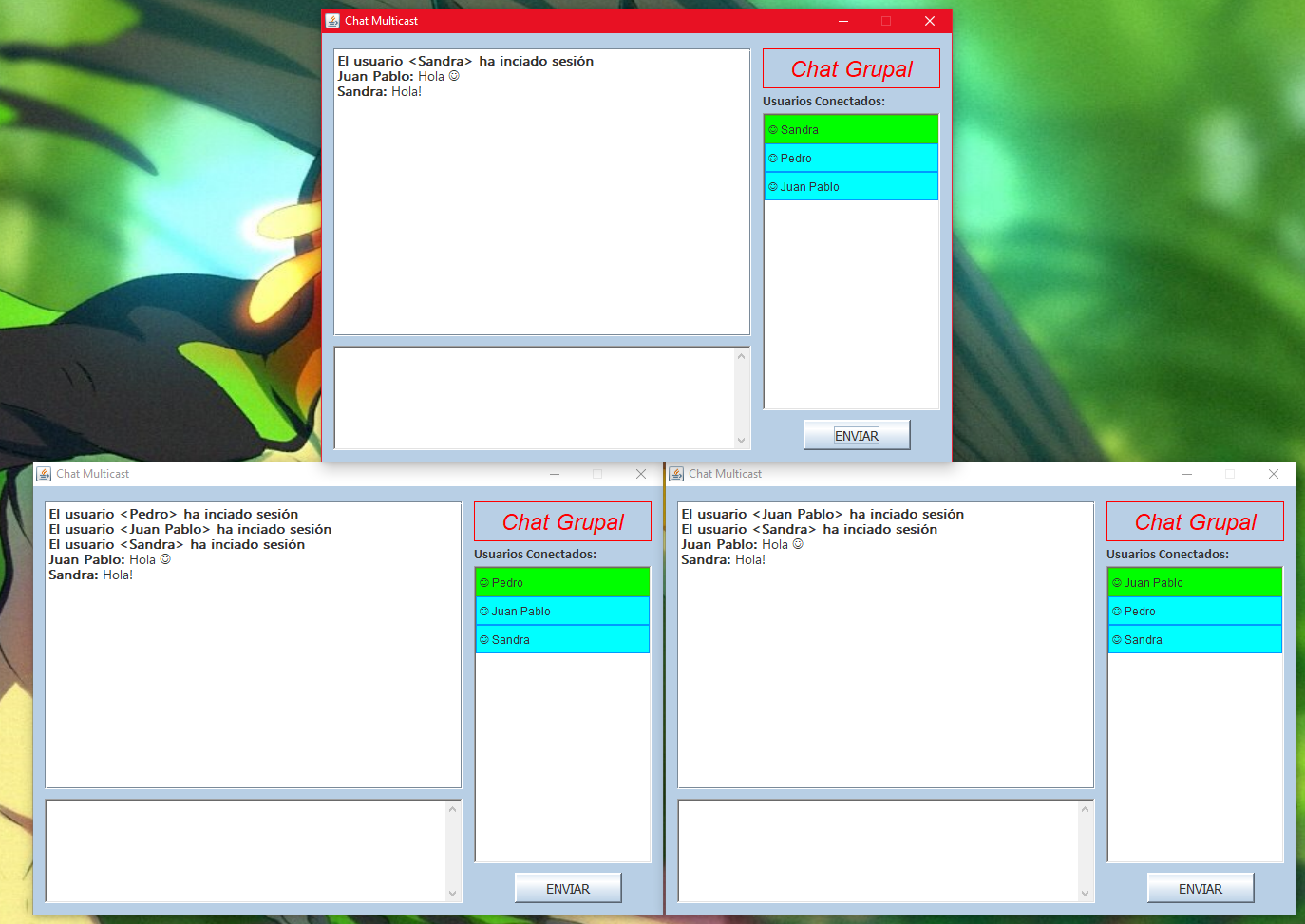


* **Inicio de sesión**



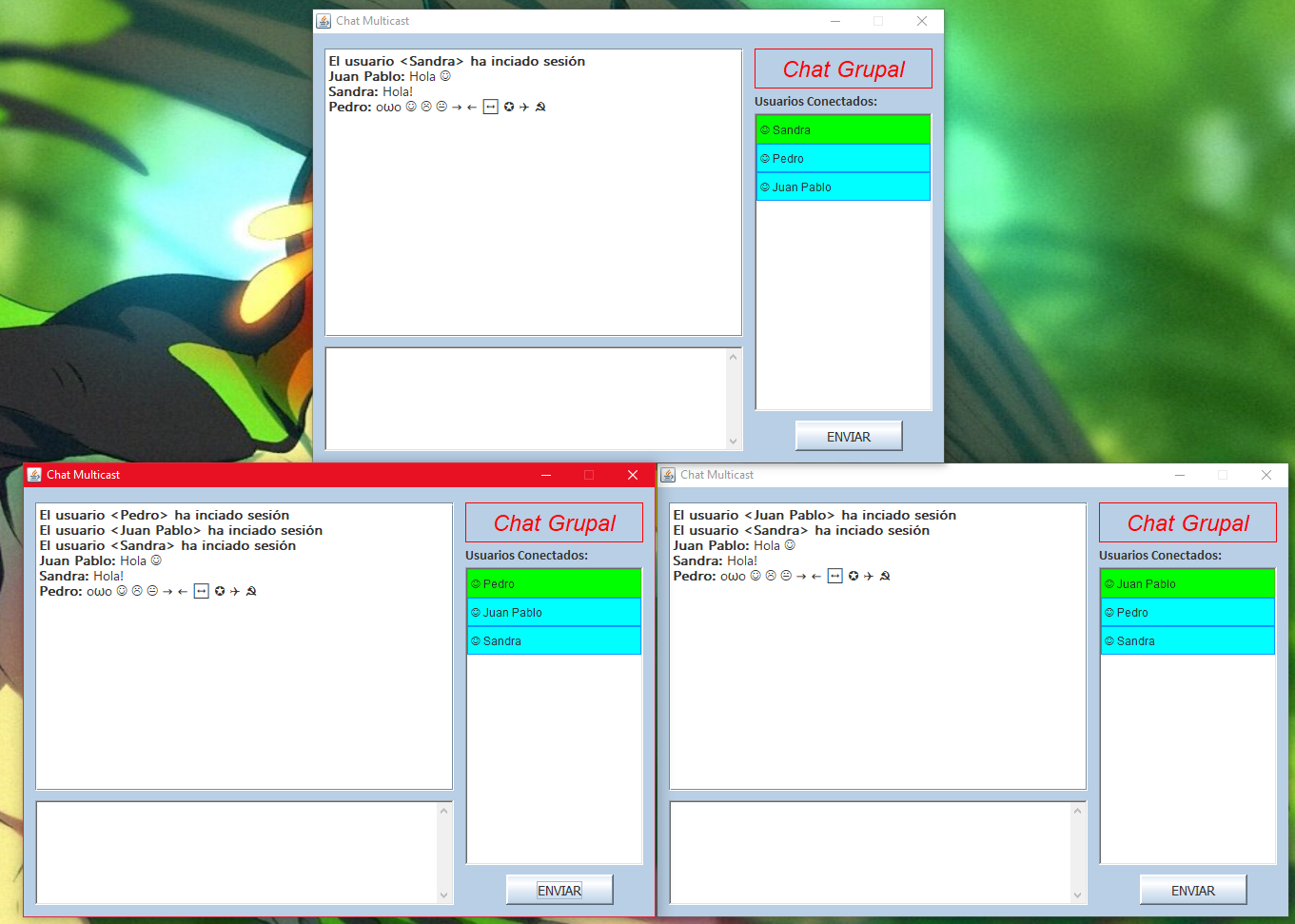


* **Mensaje Público**





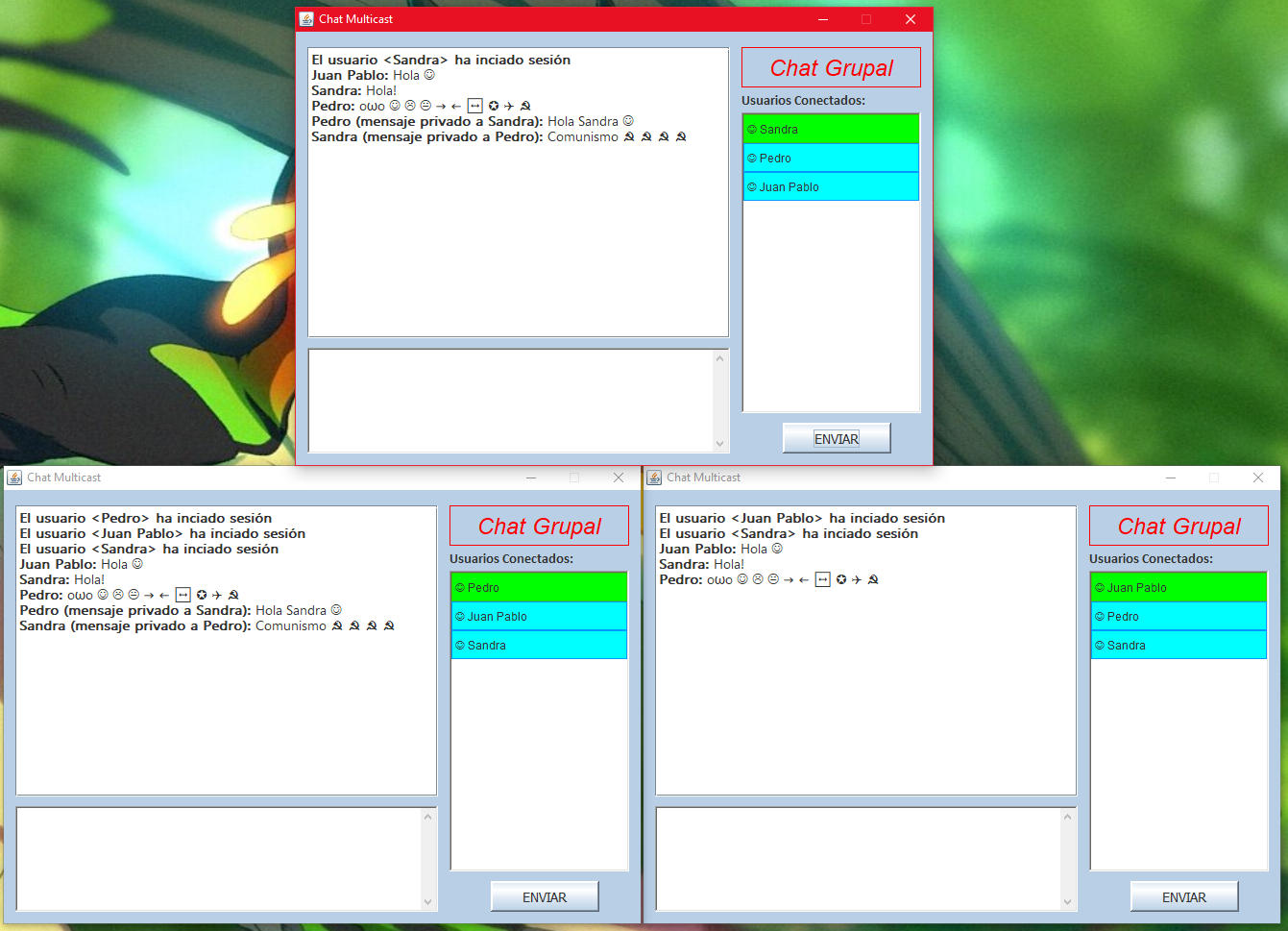
* **Emoticonos**





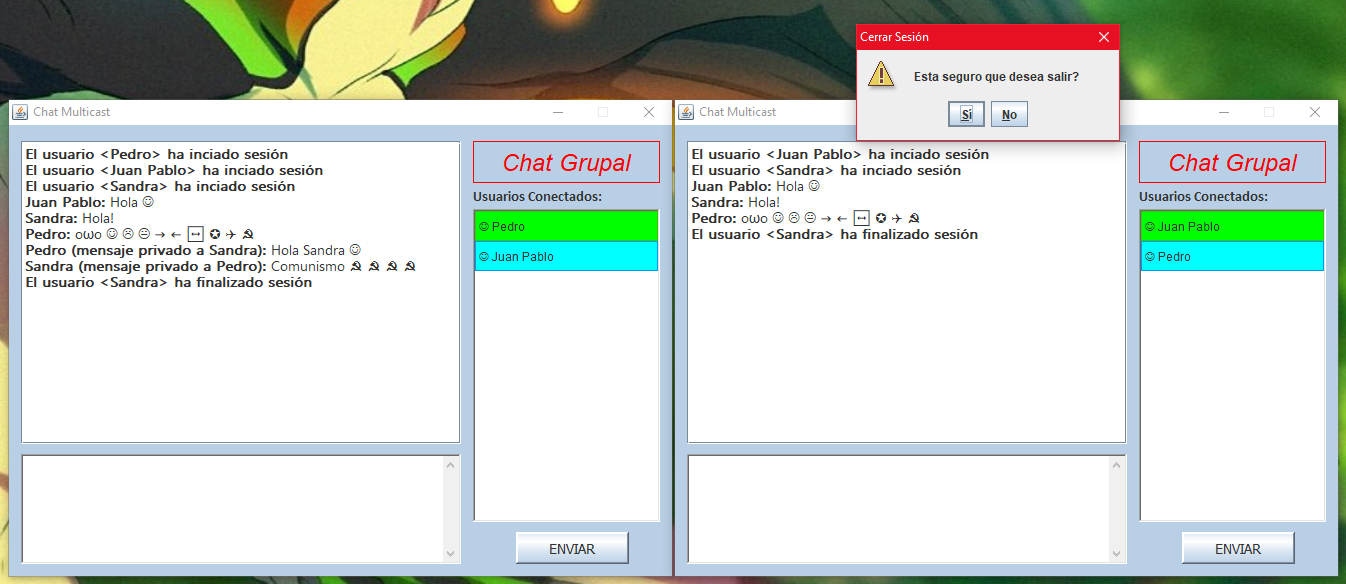
**NOTA:** Para el chat que se muestra en consola no se incluyeron loe emoticones

* **Mensaje Privado**





* **Fin de sesión**





**Conclusión:**

Los servicios de chat son una herramienta bastante útil, pues nos permiten hablar con amigos o familiares que se encuentran en distintas partes del mundo a la nuestra, para así poder mantenernos comunicados con ellos sin la necesidad de que se encuentren en nuestro mismo espacio; en especial en estos días, donde debido a la situación con el COVID-19 resulta más difícil poder visitar e interactuar con aquellos familiares que se encuentran en otros lugares distintos al nuestro.

**Bibliografía:**

* Oracle Help Center. 2020. *JDK 14 Documentation - Home*. [online] Available at: <https://docs.oracle.com/en/java/javase/14/>
* Docs.oracle.com. 2020. *Overview (Java Platform SE 7)*. [online] Available at: <https://docs.oracle.com/javase/7/docs/api/overview-summary.html>