



**INSTITUTO POLITÉCNICO NACIONAL**  
**ESCUELA SUPERIOR DE CÓMPUTO**



-----**APLICACIONES EN COMUNICACIONES EN RED**-----

**PRÁCTICA 3:**

Chat Multicast

**Alumno:**

Meza Vargas Brandon David

**Grupo:**

3CM16

**Profesor:**

Moreno Cervantes Axel Ernesto

## Índice

Introducción.....	4
Desarrollo .....	5
Server .....	5
SendPacket .....	6
serverConnect.....	6
Client.....	7
Chat.....	9
initComponents .....	10
setAudio.....	11
setPlayAudio .....	13
showAvailableAudios.....	14
initEmojis.....	15
userLeft .....	15
setCurrentUsers .....	16
addChat.....	17
getCurrentMsg .....	18
newMessage .....	18
Constants .....	20
Connection.....	22
selectNetworkInterface .....	22
despliegaInfoNIC .....	24
Pruebas de funcionamiento .....	25
Conclusiones.....	31
Bibliografía.....	32

## Índice de ilustraciones

Ilustración 1. Server hilo.....	6
Ilustración 2. Método sendPacket. ....	6
Ilustración 3. Método serverConnect. ....	7
Ilustración 4. Hilo cliente.....	9
Ilustración 5. Constructor del chat.....	10
Ilustración 6. Método initComponents. ....	11
Ilustración 7. Método setAudio.....	13
Ilustración 8. Método setPlayAudio.....	14
Ilustración 9. Método showAvalableAudios .....	14
Ilustración 10. Método initEmojis .....	15
Ilustración 11. Método userLeft.....	16
Ilustración 12. Método setCurrentUsers.....	17
Ilustración 13. Método addChat .....	18
Ilustración 14. Método getCurrentMsg .....	18
Ilustración 15. Método newMessage.....	20
Ilustración 16. Clase constants.....	22
Ilustración 17. Método selectNetworkInterface.....	24
Ilustración 18. Método despliegainfoNIC .....	24
Ilustración 19. Ejecución del servidor. ....	25
Ilustración 20. Solicitud de nombre al usuario.....	25
Ilustración 21. Solicitud de selección de interfaz de red.....	26
Ilustración 22. Interfaz del chat.....	26
Ilustración 23. Ventana de los clientes. ....	27
Ilustración 24. Mensaje general.....	27
Ilustración 25. Mensajes de usuarios.....	28
Ilustración 26. Envío de audio. ....	28
Ilustración 27. Reproduciendo audio, .....	29
Ilustración 28. Mensaje privado de brandon a liz .....	29
Ilustración 29. Mensaje de liz a brandon .....	30
Ilustración 30. Mandando emojis.....	30
Ilustración 31. Emojis en el chat. ....	30

## Introducción

Muchas empresas hacen uso de internet para ofrecer sus productos y servicios, ya que a través de este medio pueden estar en contacto directo con los clientes potenciales. Algunas de las principales ventajas de Internet como medio para realizar negocios son la cobertura a nivel global, así como la disponibilidad del servicio los 365 días, las 24 horas. del día. Sin embargo, estas ventajas se convierten también en un reto para las empresas, pues se debe proporcionar a los clientes una vía de comunicación directa para resolver las dudas de los clientes potenciales, atender sus peticiones, así como proporcionarles información adicional a la expuesta en sus portales. Existen diversos recursos que pueden ser utilizados, como los foros o el correo electrónico, pero ninguno de ellos permite la comunicación en tiempo real. El chat es un excelente recurso para este tipo de propósito, ya que además de ser una vía de comunicación en tiempo real, permite comunicar a dos o más usuarios entre sí. Típicamente los chats se han implementado haciendo uso de sockets de flujo, pero existen otras alternativas que también merece la pena probar, tales como los sockets de datagrama o los de multidifusión

Hoy en día 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 se pueden interactuar dos o más personas a la vez.

Por lo anterior, en el presente documento se presenta la práctica realizada la cual consta de un chat que permite mandar mensajes a un chat general y mensajes privados, así como emojis y audios.

## Desarrollo

A continuación se muestra el desarrollo de la práctica con las capturas de todo el código utilizado

### Server

Primeramente tenemos el servidor el cual se encargará de recibir mensajes del cliente y a su vez enviar mensajes indicando que es una respuesta del servidor

El hilo se encarga de determinar el tipo de mensaje del que se trata y realizar una acción en base a eso, si el mensaje indica un inicio de chat se le mandará a los clientes la lista de usuarios conectados, si el mensaje se trata de un mensaje tal cuál este se mandará al cliente y si se indica un fin de sesión se removerá ese cliente de los chats y se enviará a los clientes conectados la nueva lista de usuarios en el chat.

```
/**
 * Thread used to connect the server app
 * @param none
 * @return none
 */
@Override
public void run(){
    request = true; /** requesting server for the very first time
    while(true){
        try{
            /** Joining to the multicast group
            group = InetAddress.getByName(Constants.GROUP);
            s = new MulticastSocket(Constants.SERVER_PORT);
            /** Connecting to the server
            serverConnect();
            s.setSoTimeout(100); /** Server TTL
            buffer = new byte[Constants.BUFFER];
            packet = new DatagramPacket(buffer, buffer.length);
            /** Receiving message from client
            s.receive(packet);
            data = packet.getData();
            message = new String(data);

            System.out.println("Message: " + message);

            /** Determining type of message
            if(message.contains("<inicio>")){
                message = message.substring(8); /** User name being submitted
                String username = "";
                String usersS;
                /** Determining the user who just joined
                for(int i = 0; Character.isLetter(message.charAt(i)) && i < message.length(); i++) username = username + message.charAt(i);
                users.add(username);
                usersS = "<usuarios>" + users.toString(); /** Converting the array of users into a string ready to be sent
                System.out.println("Usuarios: " + usersS);
                /** Sending packet
                sendPacket(usersS, group);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        }else if(message.contains("quit")){
            /** Extracting message from the client
            message = message.substring(1);
            message = "x" + message;
            System.out.println("message server: " + message);
            sendPacket(message, group);
        }else if(message.contains("who")){
            message = message.substring(3); // User name being subtract
            String username = "";
            String users[];
            /** Determining the user who has just disconnected
            for(int i = 0; Character.isLetter(message.charAt(i)) && i < message.length(); i++) username += message.charAt(i); // Extract the username into a new string
            System.out.println("User: " + username);
            users.remove(username);
            users[] = "users[]" + users.toString(); /** Converting the array of users into a string ready to be sent
            /** Sending packet
            sendPacket(users[], group);
            try{
                Thread.sleep(1000);
            }catch (InterruptedException ie){
            }
        }
    }catch (Exception e){
        e.printStackTrace();
    }
}

```

Ilustración 1. Server hilo.

## SendPacket

Este método se encarga de enviar el paquete al cliente como se ha realizado en prácticas anteriores, a continuación se presenta el código correspondiente.

```

/**
 * Method that send a packet to the cliente
 * @param msg The message to be sent
 * @param gpo The multicast group
 */
public static void sendPacket(String msg, InetAddress gpo){
    try {
        byte[] b = msg.getBytes();
        packet = new DatagramPacket(msg.getBytes(), msg.length(), gpo, Constants.SERVER_PORT);
        System.out.println("Mensaje " + msg + " enviado con un ttl = " + s.getTimeToLive());
        s.send(packet);
        s.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Ilustración 2. Método sendPacket.

## serverConnect

Este método se encarga de mostrar en el servidor las interfaces de red disponibles para que se seleccione una y de esta manera unir al servidor al grupo multicast, como el servidor está en un ciclo infinito se le verifica si el servidor ha sido invocado solo una vez, si no es así se selecciona la misma red elegida la primera vez de manera automática.

```

/**
 * Method that make the server connection
 * @param none
 * @return none
 */
public void serverConnect(){
    try {
        if(request){
            s = new MulticastSocket(Constants.SERVER_PT0);
            Connection.selectNetworkInterface(Constants.SERVER_PT0, s, server: true, group);
            ni = s.getNetworkInterface();
            request = false;
        }else{
            SocketAddress dir;
            try{
                dir = new InetSocketAddress(Constants.GROUP, Constants.SERVER_PT0);
            }catch(Exception e){
                e.printStackTrace();
                return;
            }//catch
            s.joinGroup(dir, ni);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

*Ilustración 3. Método serverConnect.*

## Client

En el cliente tenemos un hilo que se encarga de gestionar las conexiones y mensajes de los clientes en el chat, lo que se hace primero es unir al cliente al mismo grupo multicast al que está unido el servidor.

Al inicio se le manda al servidor un mensaje de inicio de chat, pues el usuario al ejecutar el programa y al unirse adecuadamente a un grupo da inicio al chat.

Posteriormente en un ciclo infinito se pregunta por la operación que se está realizando en el cliente, si es escritura quiere decir que tenemos que escribir un mensaje, para esto obtenemos el mensaje dependiendo si es de fin de chat, mensaje general o mensaje privado entre usuarios

Si la operación que se está realizando es de lectura entonces tenemos que setear un nuevo mensaje el cual es el que se enviará.

```

public class Client extends Thread{
    private ArrayList<String> users;
    private static MulticastSocket s;
    private ObjectInputStream ois;
    private ByteArrayInputStream bais;
    private byte[] buffer;
    private InetAddress group;
    //private byte[] data;
    private String message; //Message to be sent to the server
    private static Chatss c;

    /**
     * Client thread that will be checking if is reading case o writing case
     */
    @Override
    public void run(){
        try {
            System.out.println("Cliente");
            s = new MulticastSocket(Constants.SERVER_PTD);
            String userName = c.getUserName();
            group = InetAddress.getByName(Constants.GROUP);
            /** Joining to the multicast group
             Connection.selectNetworkInterface(Constants.SERVER_PTD, s,  server false, group);

            //String ini = String.format(Constants.USER_JOINED_FORMAT, userName);
            /** Indicating the user connection
            String ini = "<inicio>" + userName;
            System.out.println("bytes " + ini.getBytes() + " length " + ini.length());
            //System.out.println("bytes " + ini2.getBytes() + " length " + ini2.length());
            DatagramPacket packet = new DatagramPacket(ini.getBytes(), ini.length(), group, Constants.SERVER_PTD);
            /** Sending the client who has just joined
            s.send(packet);

```



```

    /** For ever
    for(;;){
        /** If operation is equals 1 means that it'll write
        if(c.getOp() == 1){
            /** If user is disconnected
            if(!c.isConnected()){
                System.err.println("Conectado igual: " + c.isConnected());
                message = String.format(Constants.USER_DISCONNECTED_FORMAT, userName);
            }else{
                /** If is the Tab 0 means that the message is to the general room
                if(c.getTab() == 0){
                    message = String.format(Constants.MESSAGE_FORMAT, userName, c.getCurrentMsg());
                }else{
                    /** If tab is not 0 the message is a private one
                    message = String.format(Constants.PRIVATE_MESSAGE_FORMAT, userName, c.getReceiver(c.getTab()), c.getCurrentMsg());
                }
            }
            /** Sending the message either a general or a private message
            packet = new DatagramPacket(message.getBytes(), message.length(), group, Constants.SERVER_PORT);
            s.send(packet);
            c.setOp(0);

        }else if(c.getOp() == 0){ /** We have to read
            s.setSoTimeout(100);
            try {
                byte[] data = new byte[Constants.SUFFER];
                DatagramPacket packetr = new DatagramPacket(data, data.length);
                s.receive(packetr);
                byte[] b = packetr.getData();
                String nm = new String(b);
                System.err.println("Mensaje" + nm + " recibido con un ttl = " + s.getTimeToLive());
                c.newMessage(nm);
            } catch (Exception e) {
            }
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
}

```

Ilustración 4. Hilo cliente.

## Chat

En esta parte tenemos todo lo necesario para desplegar la interfaz gráfica del usuario

```

public Chatss(int op){
    operation = op;
    /* Asking username
    userName = JOptionPane.showInputDialog( parentComponent: null, Constants.USERNAME);
    /* Centring window
    Toolkit myScreen = Toolkit.getDefaultToolkit();
    Dimension screenSize = myScreen.getScreenSize();
    connected = true;

    window = new JFrame();
    window.setSize( width: (screenSize.width / 2)+100, height: screenSize.height / 2);
    window.setLocation( x: screenSize.width / 4, y: screenSize.height / 4);
    System.out.println(screenSize.width / 2 + "altuura: " + screenSize.height / 2);
    window.setResizable(false);
    window.setTitle(String.format(Constants.WINDOW_TITLE, userName));
    window.setLocationRelativeTo(null);
    window.setDefaultCloseOperation(EXIT_ON_CLOSE);
    userLeft();

    tabs = new JTabbedPane();
    emojisPanel = new JPanel(new GridLayout( rows: 5, cols: 4));
    currentUsers = new ArrayList();
    currentUsersString = new ArrayList();
    privateChats = new ArrayList();
    chatsInTabs = new ArrayList();
    initComponents();
    initEmojis();
    window.setVisible(true);
}

```

*Ilustración 5. Constructor del chat*

## initComponents

Aquí empezamos por inicializar los componentes participantes en la interfaz del usuario, entre ellos la ventana principal y el botón de enviar, así como la llamada a otros métodos que inicializan otras cosas.

```

/**
 * Method that initialize the client GUI components
 */
public void initComponents(){
    tabs.setBounds( x: 5, y: 10, width: 750, height: 450);
    JLabel usersSection = new JLabel(Constants.CURRENT_USERS);

    panel = new JPanel();
    panel.setLayout(null);
    window.getContentPane().add(panel);
    panel.setBackground(new Color( r: 255, g: 246, b: 234));
    usersSection.setBounds( x: 830, y: 10, width: 200, height: 10);
    panel.add(usersSection);
    addChat( title: "Sala general");
    panel.add(tabs);
    btn = new JButton( text: "Enviar");

    btn.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent ae){
            if(ae.getSource() == btn){
                operation = 1;
                System.out.println("Opeación pulsado " + operation);
            }
        }
    });

    btn.setBounds( x: 475, y: 465, width: 120, height: 40);
    btn.setBackground(Color.WHITE);
    btn.setFont(new FontUIResource( name: "Roboto", Font.BOLD, size: 20));
    panel.add(btn);
    chat = new JTextField();
    chat.setBounds( x: 5, y: 465, width: 450, height: 40);
    panel.add(chat);

    setAudio();
    setPlayAudio();
    setCurrentUsers();
}

```

*Ilustración 6. Método initComponents.*

## setAudio

Aquí se crean los componentes para enviar el audio, se crea el botón correspondiente y su event listener que se encargará de enviar el audio primeramente se encarga de establecer la carpeta donde se guardará el audio y posteriormente se crea una instancia de la clase que se encarga de grabar el audio, se grabará por 5 segundos

```

/**
 * Method that set the Audio components
 */

public void setAudio(){
    JButton audio = new JButton( text: "Enviar audio");
    audio.setBounds( x: 610, y: 465, width: 120, height: 40);
    audio.setBackground(Color.WHITE);
    audio.setFont(new FontUIResource( name: "Roboto", Font.BOLD, size: 15));

    audio.addActionListener(new ActionListener(){

        /**
         * The send audio button action listener will send the audio
         * @param ae the action event
         */
        @Override
        public void actionPerformed(ActionEvent ae){
            if(ae.getSource() == audio){
                chat.setText("");
                chat.setText("Audio enviado");
                operation = 1;
                DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yyyy-HH-mm-ss");
                LocalDateTime now = LocalDateTime.now();
                String audioName = String.format(Constants.AUDIO_NAME, dtf.format(now), userName);
                final JavaSoundRecorder recorder;
                int tab = tabs.getSelectedIndex();
                if(tab == 0){
                    recorder = new JavaSoundRecorder( path: "Audios/General", audioName, isGeneral: true);
                }else{
                    if(userName.equals(reciever)){
                        path = String.format(Constants.PATH_AUDIO_FORMAT, userName, chatsInTabs.get(tab));
                    }else{
                        path = String.format(Constants.PATH_AUDIO_FORMAT, chatsInTabs.get(tab), userName );
                    }
                    recorder = new JavaSoundRecorder(path, audioName, isGeneral: false);
                }
            }
        }
    })
}

```

```

try{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    //Get and display a list of
    // available mixers.
    Mixer.Info[] mixerInfo = AudioSystem.getMixerInfo();
    int micro = 5;
    // creates a new thread that waits for a specified
    // of time before stopping
    Thread stopper = new Thread(new Runnable() {
        public void run() {
            try {
                Thread.sleep(Constants.RECORD_TIME);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
            recorder.finish();
        }
    });
    stopper.start();
    // start recording
    recorder.start(micro);
    br.close();
} catch (Exception e) {
    e.printStackTrace();
} //catch
}
});
panel.add(audio);

```

*Ilustración 7. Método setAudio*

## setPlayAudio

Este método crea el botón para reproducir un audio, posteriormente se manda a llamar la clase que muestra los audios disponibles.

```

/**
 * Method that set the play audio button and its components
 */
public void setPlayAudio(){
    JButton playAudio = new JButton( text: "Reproducir audio");
    playAudio.setBounds( x: 750, y: 465, width: 180, height: 40);
    playAudio.setBackground(Color.WHITE);
    playAudio.setFont(new FontUIResource( name: "Roboto", Font.BOLD, size: 14));
    playAudio.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent ae){
            if(ae.getSource() == playAudio){
                int tab = tabs.getSelectedIndex();
                if(tab == 0){
                    showAvailableAudios( path: "Audios/General");
                }else{
                    showAvailableAudios(String.format(path));
                }
            }
        }
    });
    panel.add(playAudio);
}

```

Ilustración 8. Método setPlayAudio

## showAvailableAudios

Este método se encarga de ir a la carpeta correspondiente para mostrarle al usuario los audios que están disponibles para que el pueda seleccionar uno y a partir de eso se crea una instancia de la clase que se encarga de reproducir el audio seleccionado.

```

/**
 * Show the available audios which the client can select to reproduce
 * @param path The directory path where the audios are saved
 * @return none
 */
public void showAvailableAudios(String path){
    PlayWavFile ps;
    File[] audios;
    audios = (new File(path)).listFiles();
    if(audios.length == 0) JOptionPane.showMessageDialog( JOptionPane.NULL, message: "No hay audios para reproducir", title: "Sin audios", JOptionPane.ERROR_MESSAGE);
    else{
        File audioSelected = (File) JOptionPane.showInputDialog( JOptionPane.NULL, message: "Reproducir un audio", title: "Selecciona el audio a reproducir",
            JOptionPane.QUESTION_MESSAGE, UIManager.getIcon( Icon: "OptionPane.questionIcon"), audios, audios[0]);
        ps = new PlayWavFile(audioSelected.getAbsolutePath());
        ps.play();
    }
}

```

Ilustración 9. Método showAvailableAudios



## initEmojis

Este método se encarga de poner los emojis disponibles en la interfaz de usuario. Cada emoji corresponde a un botón que cuando el usuario lo pulsa se le adjunta a su mensaje de texto.

```
/**
 * Method that sets the emojis components
 */
public void initEmojis(){
    emojisPanel.setBounds( x: 780, y: 360, width: 240, height: 70);
    Constants.EMOJIS_LIST.forEach((k,e) -> {
        JButton emojiBtn = new JButton(k);
        System.out.println(k);
        emojisPanel.add(emojiBtn);
        emojiBtn.setBackground(Color.WHITE);
        emojiBtn.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent ae){
                if(ae.getSource() == emojiBtn){
                    chat.setText(chat.getText() + e);
                }
            }
        });
    });

    JLabel emojisText = new JLabel( text: "EMOJIS");
    emojisText.setBounds( x: 780, y: 310, width: 240, height: 70);
    panel.add(emojisPanel);
    panel.add(emojisText);
}
```

Ilustración 10. Método initEmojis

## userLeft

Este método indica cuando un usuario se desconecta del chat, este método agrega un escuchador a la ventana al momento de que se pulsa el botón de cerrar

Cuando pasa eso se cambió el estado de conectado a false y se cambia la operación a escritura para indicar que salió del chat.

```

/**
 * Method that indicates when a user leaves the chat
 */
public void userLeft(){
    window.addWindowListener((WindowAdapter) windowClosing(e) → {
        connected = false;
        operation = 1;
        currentUsersString.remove(userName);
        setCurrentUsers();
        System.out.println("Saliendo");
        try {
            Thread.sleep( millis: 1000);
        } catch (InterruptedException ie) {
            System.err.println(ie.getMessage());
        }
    });
}

```

*Ilustración 11. Método userLeft*

### **setCurrentUsers**

Este método recorre el arreglo que contiene los usuarios conectados en tiempo real, poniéndolos todos en un botón para que el usuario al hacer clic en uno de ellos puedan iniciar un chat privado. A cada botón se le asigna un escuchador que agregará al usuario clicando en una nueva pestaña del tabbed Pane

Al final se tiene que repintar la ventana para ver cuando hay actualizaciones en los usuarios conectados.



```

/**
 * Method that set the current Users in the chat
 */
public void setCurrentUsers(){
    int i = 0;
    System.err.println("Dentro setCurrentUsers");
    for(String user: currentUsersString){
        if(!user.equals(userName)){
            JButton userBtn = new JButton(user);
            currentUsers.add(userBtn);

            userBtn.setBounds( x: 800, y: 35+i, width: 200, height: 25);
            i += 35;
            userBtn.setName(user);
            userBtn.setBackground(new Color( r: 255, g: 238, b: 238));
            userBtn.setFont(new FontUIResource( name: "Roboto", Font.BOLD, size: 15));
            userBtn.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent ae){
                    if(ae.getSource() == userBtn){
                        System.out.println("Chat privado con " + user);
                        if(chatsInTabs.contains(user)) tabs.setSelectedIndex(chatsInTabs.indexOf(user));
                        else{
                            addChat(user);
                            tabs.setSelectedIndex(chatsInTabs.indexOf(user));
                        }
                    }
                }
            });
            panel.add(userBtn);
            System.err.println(user);
            window.repaint();
        }
    }
}
}

```

*Ilustración 12. Método setCurrentUsers*

## addChat

Aquí simplemente se agrega a las pestañas una nueva en caso de que sea un nuevo chat se agrega una nueva ventana con un título que será igual al nombre del usuario con el que se inició un chat.

```

/**
 * Method that add a new chat to the tabbed pane
 * @param title The tab name
 */

public void addChat(String title){
    JPanel chat = new JPanel();
    chat.setLayout(null);
    tabs.addTab(title, chat);
    JEditorPane ep = new JEditorPane();
    JTextArea chatArea = new JTextArea();
    chatArea.setForeground(Color.BLACK);
    chatArea.setEditable(false);
    privateChats.add(chatArea);
    /* Adding chat o a scroll pane
    scroll = new JScrollPane(privateChats.get(privateChats.size() - 1));
    scroll.setBounds( x: 5, y: 5, width: 745, height: 445);
    chatsInTabs.add(title);
    chat.add(scroll);
}

```

*Ilustración 13. Método addChat*

### getCurrentMsg

Este método solo se encarga de devolver el mensaje actual.

```

public String getCurrentMsg(){
    currentMessage = chat.getText();
    chat.setText("");
    return currentMessage;
}

```

*Ilustración 14. Método getCurrentMsg*

### newMessage

Este método es el más importante de la clase pues se encarga de establecer de que tipo de mensaje se trata y crear el mensaje.

En primer lugar tenemos que si es un mensaje que indica un inicio de sesión se setea la lista de usuarios conectados.

Si es un mensaje se corta la parte que indica que es un mensaje y se verifica si es un mensaje privado, de ser así se corta el mensaje para determinar el receptor y emisor. Si el

usuario conectado es igual al que recibe el mensaje y el que mando el mensaje ya está en las pestañas del chat solo se cambia la pestaña actual, en caso contrario se crea la pestaña con el método antes mencionado. En caso de que el usuario conectado sea igual al que manda el mensaje se obtiene la pestaña y si no corresponde a un chat privado se obtiene la pestaña 0 que corresponde al chat general.

```
/**
 * Method that will set the new message depending if it a private or general message
 * @param m The message received
 */
public void newMessage(String m){
    System.out.println("Mensaje recibido: " + m);
    if(m.contains("<usuario>")){
        m = m.substring(10);
        String [] contacts = m.split(Regex.quote(""));
        currentUsersString.clear();
        currentUsers.clear();
        //Arrays.stream(contacts).forEach(System.out::println);
        Arrays.stream(contacts).forEach(c -> currentUsersString.add(c.replaceAll(Regex.quote("[^a-zA-Z]", ""))));
        System.out.println(Constants.CURRENT_USERS);
        for(String c :currentUsersString ){
            System.out.println(c);
        }
        System.err.println("Setting current users");
        setCurrentUsers();
    }else if(m.startsWith("<=>")){
        m = m.substring(0);
        m = m.replace(Regex.quote("<"), replacement: "\\u083D\\u0E08");
        m = m.replace(Regex.quote("<D"), replacement: "\\u083D\\u0E04");
        m = m.replace(Regex.quote("<TD"), replacement: "\\u083D\\u0E02");
        m = m.replace(Regex.quote("<D>"), replacement: "\\u083D\\u0E07");
        m = m.replace(Regex.quote("<3"), replacement: "\\u083D\\u0E0D");
        m = m.replace(Regex.quote("<p"), replacement: "\\u083D\\u0E1B");
        m = m.replace(Regex.quote("<8"), replacement: "\\u083D\\u0E0E");
        m = m.replace(Regex.quote("</"), replacement: "\\u083D\\u0E15");
        m = m.replace(Regex.quote("<("), replacement: "\\u083D\\u0E31");
        m = m.replace(Regex.quote("<@"), replacement: "\\u083D\\u0E2E");
        m = m.replace(Regex.quote("<("), replacement: "\\u083D\\u0E22");
        m = m.replace(Regex.quote("<:"), replacement: "\\u083D\\u0E21");
        m = m.replace(Regex.quote("<X"), replacement: "\\u083D\\u0C88");
        m = m.replace(Regex.quote("<D"), replacement: "\\u083D\\u0C7D");
        m = m.replace(Regex.quote("<kiss"), replacement: "\\u083D\\u0CB8");
        m = m.replace(Regex.quote("<+5"), replacement: "\\u083D\\u0C96");
        m = m.replace(Regex.quote("<+"), replacement: "\\u083D\\u0CA5");
    }
}
```

```

    m = m.replace(Regex("\\[\\]", replacement: "\\003E\\003A");
    m = m.replace(Regex("<", replacement: "\\003D\\00C4B");
    m = m.replace(Regex(">", replacement: "\\003D\\00C4D");
    if(m.contains("<private>")){
        sender = "";
        receiver = "";
        m = m.substring(10);
        int i = 0;
        for(i = 0; Character.isLetter(m.charAt(i));i++) sender = sender + m.charAt(i);

        m = m.substring(i+2);
        i = 0;
        for(i = 0; Character.isLetter(m.charAt(i));i++) receiver = receiver + m.charAt(i);
        m = m.substring(i+1);
        System.out.println("[" + sender + " <-> " + receiver + " <-> " + m);
        if(userName.equals(receiver)){
            if(chatsInTabs.contains(sender)){
                int tab = chatsInTabs.indexOf(sender);
                privateChats.get(tab).setText(String.format(Constants.TEXT_AREA_MESSAGE, privateChats.get(tab).getText(), sender, m));
            }else{
                addChat(sender);
                tabs.setSelectedIndex(chatsInTabs.indexOf(sender));
                int tab = tabs.getSelectedIndex();

                privateChats.get(tab).setText(String.format(Constants.TEXT_AREA_MESSAGE, privateChats.get(tab).getText(), sender, m));
            }
        }else if(userName.equals(sender)){
            int tab = tabs.getSelectedIndex();
            privateChats.get(tab).setText(String.format(Constants.TEXT_AREA_MESSAGE, privateChats.get(tab).getText(), sender, m));
        }
        else{
            privateChats.get(0).setText(privateChats.get(0).getText() + "\\n" + m);
        }
    }else if(m.contains("<init>") || m.contains("<fin>")){
        privateChats.get(0).setText(privateChats.get(0).getText() + "\\n" + m);
    }
}

```

Ilustración 15. Método newMessage

## Constants

En esta clase solo se guardan los mensajes y algunas configuraciones usadas en todo el programa como constantes.

```

public class Constants {

    /**Messages
    public static final String USER_CONNECTED = "Se ha conectado el usuario %s\n";
    public static final String USER_DISCONNECTED = "Se ha desconectado el usuario %s\n";
    public static final String CURRENT_USERS = "Usuarios conectados";
    public static final String USER_JOINED_FORMAT = "<inicio>%s";
    public static final String USER_DISCONNECTED_FORMAT = "<fin>%s";
    public static final String MESSAGE_FORMAT = "c<msg><%s>%s";
    public static final String PRIVATE_MESSAGE_FORMAT = "c<msg><privado><%s><%s>%s";
    public static final String USERNAME = "Ingresa un nombre de usuario para entrar al chat";
    public static final String WINDOW_TITLE = "Chat Multicast - %s";
    public static final String TEXT_AREA_MESSAGE = "%s\n%s: %s";

    /**Connection
    public static final String GROUP = "230.1.1.1";
    public static final int CLIENT_PTO = 9931;
    public static final int SERVER_PTO = 9930;
    public static final int BUFFER = 2048;

    /** Errors
    public static final String JOIN_GROUP_ERROR = "No fue posible unirse al grupo";
    public static final String SERVER_ERROR = "No fue posible conectarse al servidor";

    /**Emojis
    public static final Map<String,String> EMOJIS_LIST = Collections.unmodifiableMap(
        new HashMap<String,String>(){
            put("😊", ":)");
            put("😬", ":D");
            put("😬", ":'D");
            put("😬", "0:>");
            put("😬", "<3");
            put("😬", ":p");
            put("😬", "8");
            put("😬", ":/");
            put("😬", ":(");
            put("😬", ":o");
        }
    );
}

```

```

        put("😁", ":(");
        put("😞", ">");
        put("😈", "X");
        put("😱", "^^");
        put("😘", ":Kiss");
        put("💕", "<*3");
        put("💎", "**");
        put("👊", "^^");
        put("👉", "->");
        put("👎", "(Y)");
    }
}

);

/* Audio

public static final long RECORD_TIME = 5000;
public static final String PATH_AUDIO_FORMAT = "Audios/%s - %s";
public static final String AUDIO_NAME = "%s - %s.wav";

```

*Ilustración 16. Clase constants*

## Connection

En esta clase encontramos los métodos referentes a conexiones usadas por el servidor y cliente

### **selectNetworkInterface**

Este método selecciona las interfaces de red disponibles en el equipo.

Si el usuario es el servidor se le muestra la información de manera textual, en caso de que sea el cliente como se está ejecutando con una interfaz también se le da a escoger la interfaz de red de manera gráfica.

```

/**
 * Method that will join the user to the group depending on the NI choosen
 * @param pto The server port
 * @param s The multicastSocket
 * @param server Wheter if is the server or a client
 * @param group The multicast group
 */

public static void selectNetworkInterface(int pto, MulticastSocket s, boolean server, InetAddress group){
    if(server){
        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            Enumeration<NetworkInterface> nets = NetworkInterface.getNetworkInterfaces();
            int z=0;

            for (NetworkInterface netint : Collections.list(nets)){
                if(netint.supportsMulticast()){
                    System.out.print("[Interfaz "+ ++z +"]:");
                    despliegaInfoNIC(netint);
                }
            }
        } //for

        System.out.print("\nElige la interfaz multicast:");
        int interfaz = Integer.parseInt(br.readLine());
        //NetworkInterface ni = NetworkInterface.getByIndex(interfaz);
        NetworkInterface ni = NetworkInterface.getByIndex(interfaz);
        br.close();
        System.out.println("\nElegiste "+ni.getDisplayName());

        SocketAddress dir;
        try{
            dir = new InetSocketAddress(group,pto);
        }catch(Exception e){
            e.printStackTrace();
            return;
        } //catch
        s.joinGroup(dir, ni);
    }
}

```



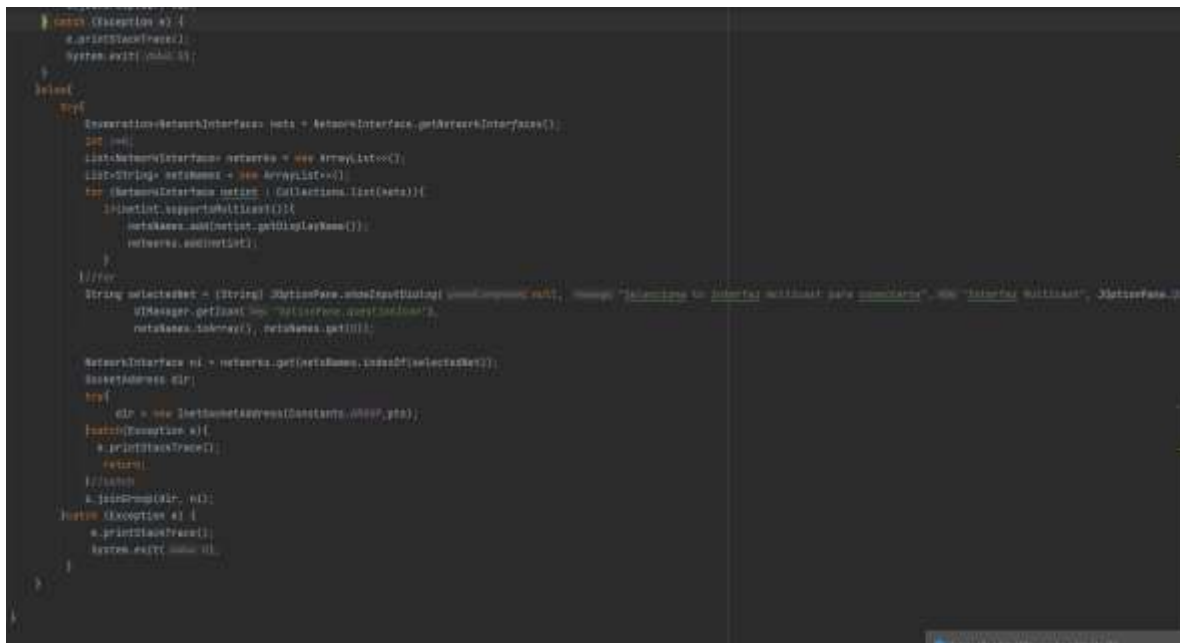


Ilustración 17. Método selectNetworkInterface

## despliegaInfoNIC

Este método solo muestra la información de la NIC como su nombre, dirección y algo muy importante, si soporta multicast o no.

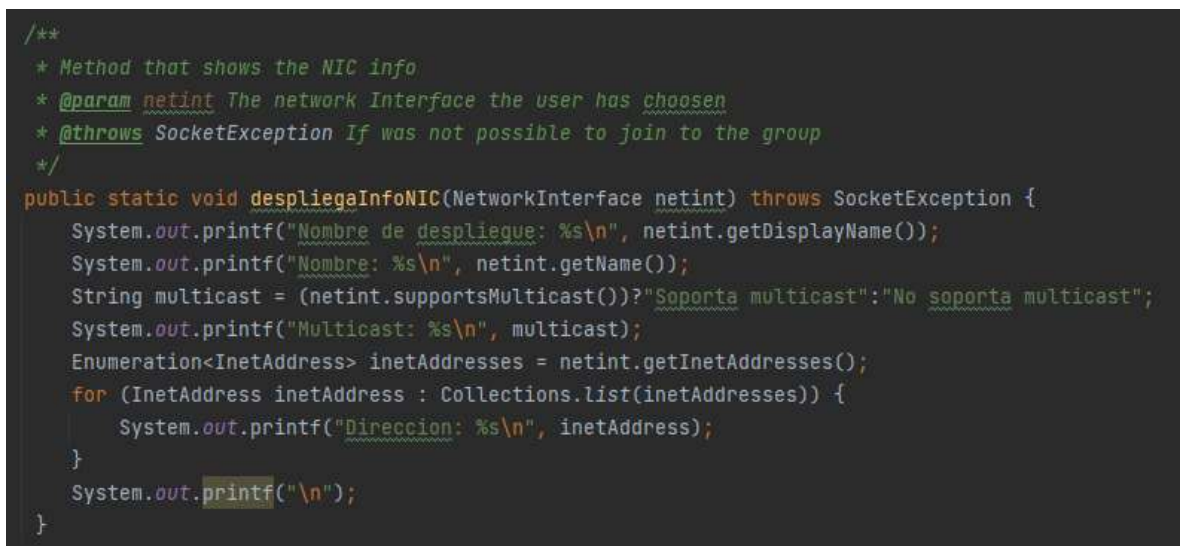


Ilustración 18. Método despliegaInfoNIC



## Pruebas de funcionamiento

Ahora pasaremos con las pruebas de la aplicación.

Primeramente debemos correr el servidor y escoger la interfaz multicast deseada para comenzar a aceptar conexiones de clientes como se muestra en la siguiente ilustración.

```
[Interfaz 45]:Nombre de despliegue: Microsoft Wi-Fi Direct Virtual Adapter #4-WFP 802.3 MAC Layer LightWeight Filter-0000
Nombre: wlan15
Multicast: Soporta multicast

[Interfaz 46]:Nombre de despliegue: Microsoft Wi-Fi Direct Virtual Adapter-WFP Native MAC Layer LightWeight Filter-0000
Nombre: wlan16
Multicast: Soporta multicast

[Interfaz 47]:Nombre de despliegue: Microsoft Wi-Fi Direct Virtual Adapter-Native WiFi Filter Driver-0000
Nombre: wlan17
Multicast: Soporta multicast

[Interfaz 48]:Nombre de despliegue: Microsoft Wi-Fi Direct Virtual Adapter-VirtualBox NDIS Light-Weight Filter-0000
Nombre: wlan18
Multicast: Soporta multicast

[Interfaz 49]:Nombre de despliegue: Microsoft Wi-Fi Direct Virtual Adapter-QoS Packet Scheduler-0000
Nombre: wlan19
Multicast: Soporta multicast

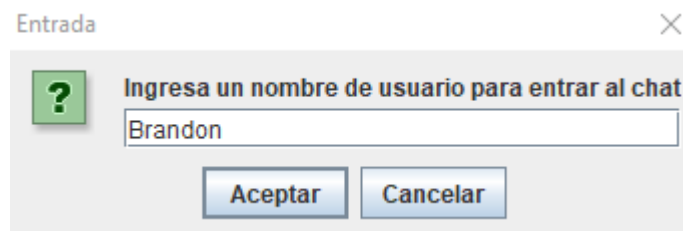
[Interfaz 50]:Nombre de despliegue: Microsoft Wi-Fi Direct Virtual Adapter-WFP 802.3 MAC Layer LightWeight Filter-0000
Nombre: wlan20
Multicast: Soporta multicast

Elige la interfaz multicast:1

Elegiste Software Loopback Interface 1
```

*Ilustración 19. Ejecución del servidor.*

Posteriormente ya podemos conectarnos como clientes, al ejecutar un cliente se le pedirá que ingrese su nombre y seleccione su interfaz de red pero ahora de una manera gráfica como se muestra a continuación.



Entrada

?

Ingresa un nombre de usuario para entrar al chat

Brandon

Aceptar Cancelar

*Ilustración 20. Solicitud de nombre al usuario*

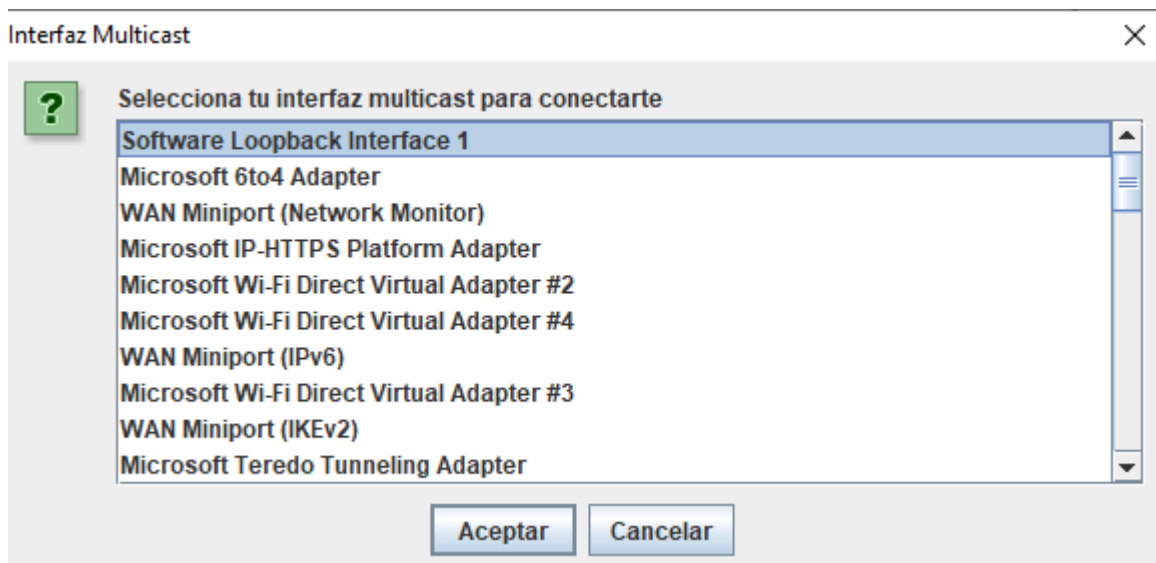


Ilustración 21. Solicitud de selección de interfaz de red.

Una vez ingresado su nombre y seleccionado su interfaz de red se abre el chat.



Ilustración 22. Interfaz del chat.

Se procederá a conectar a dos usuarios más.

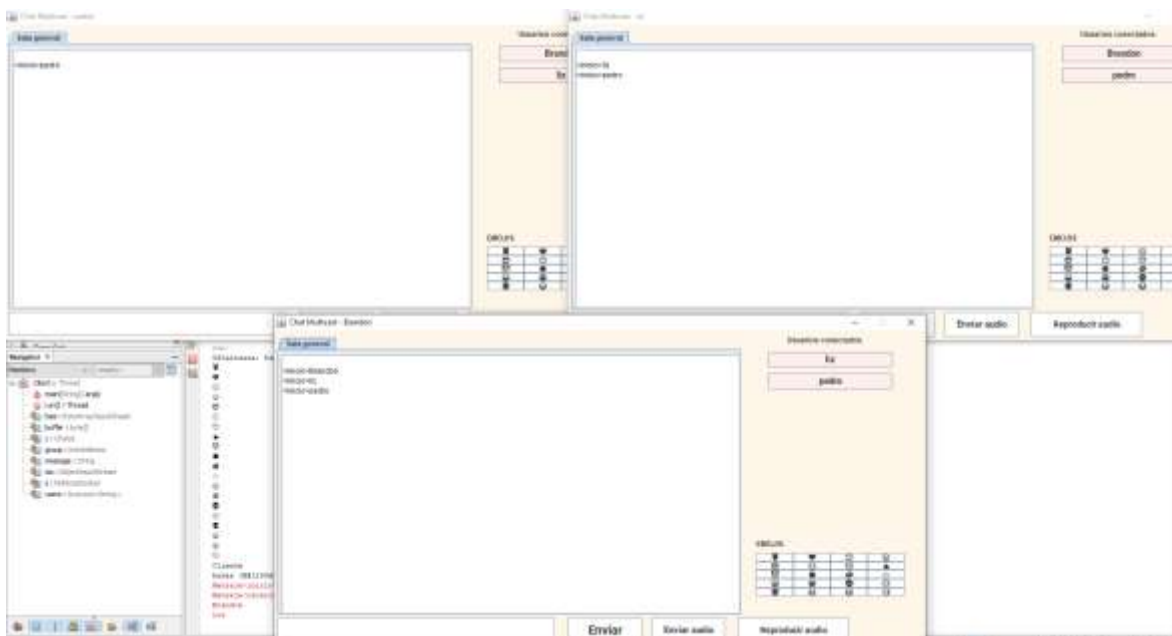


Ilustración 23. Ventana de los clientes.

Podemos ver que a cada cliente conectado se le muestran los usuarios conectados al chat en la parte de la derecha, si manda un mensaje en la sala general se manda el mensaje a todos como se muestra a continuación.

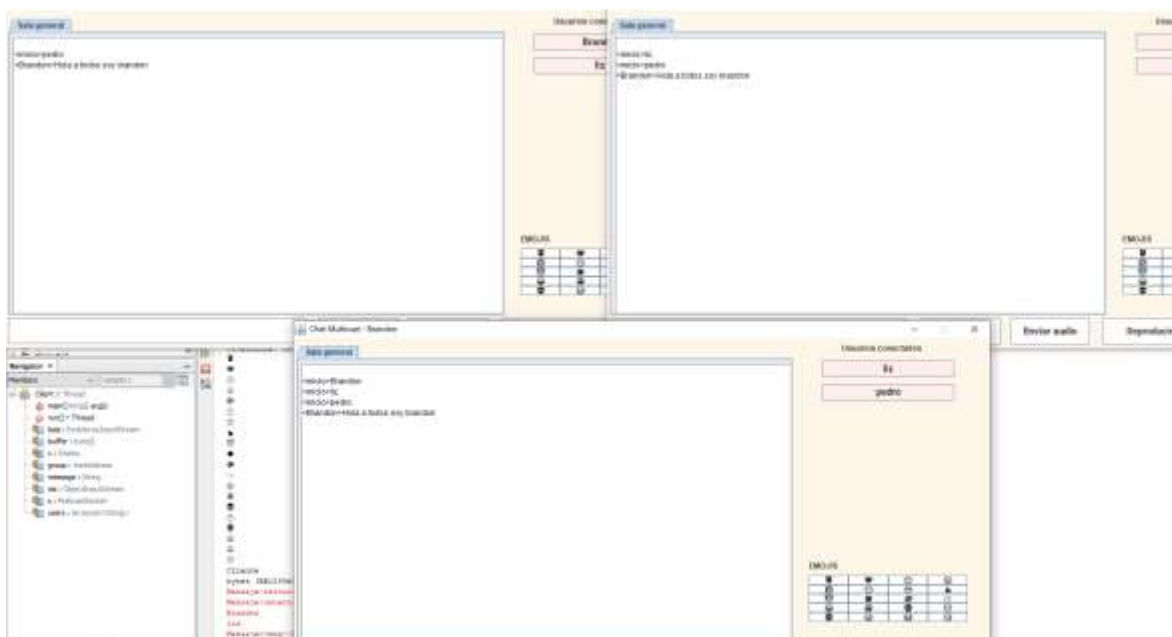


Ilustración 24. Mensaje general

Y así se pueden hacer la conversación en la sala general con los miembros activos del chat.

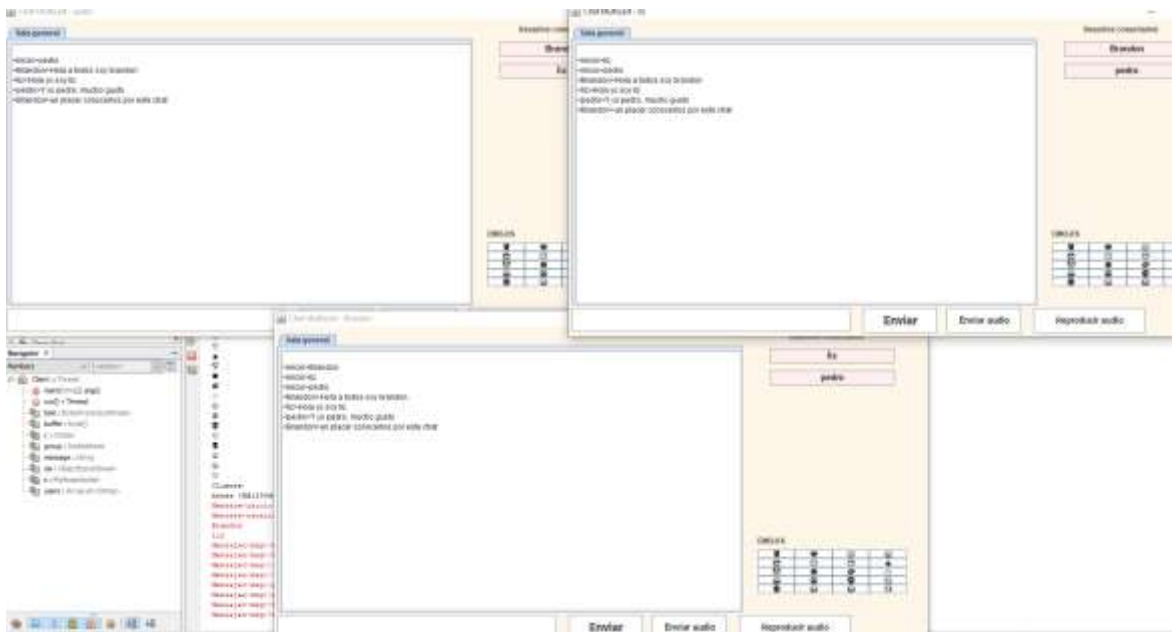


Ilustración 25. Mensajes de usuarios.

Para mandar un audio basta con pulsar el botón de enviar audio, se grabará un audio de 5 segundos y se indicará en el chat que un usuario envió un audio.

Chat Multicast - Brandon

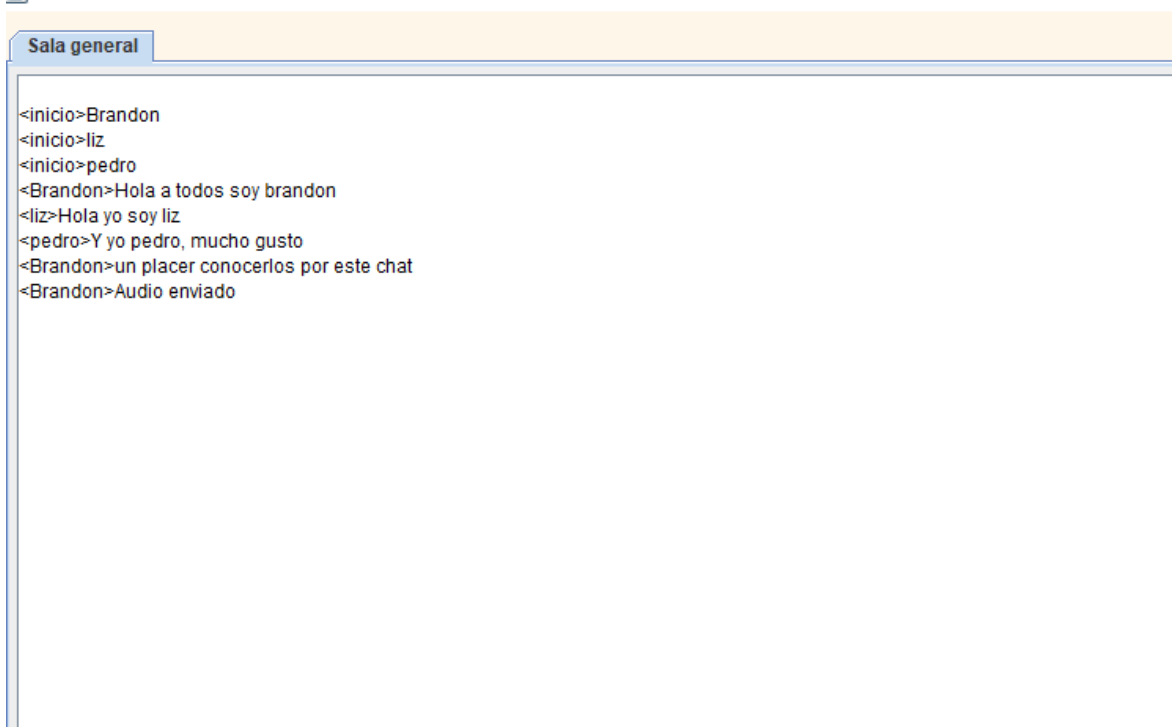
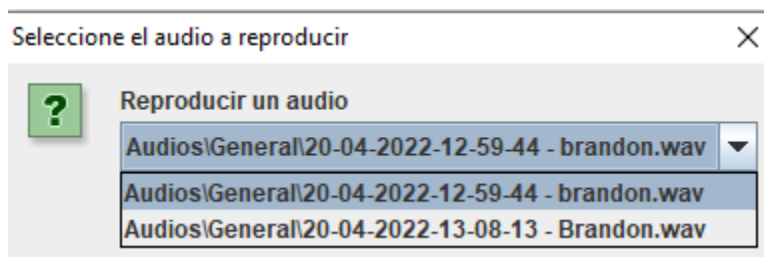


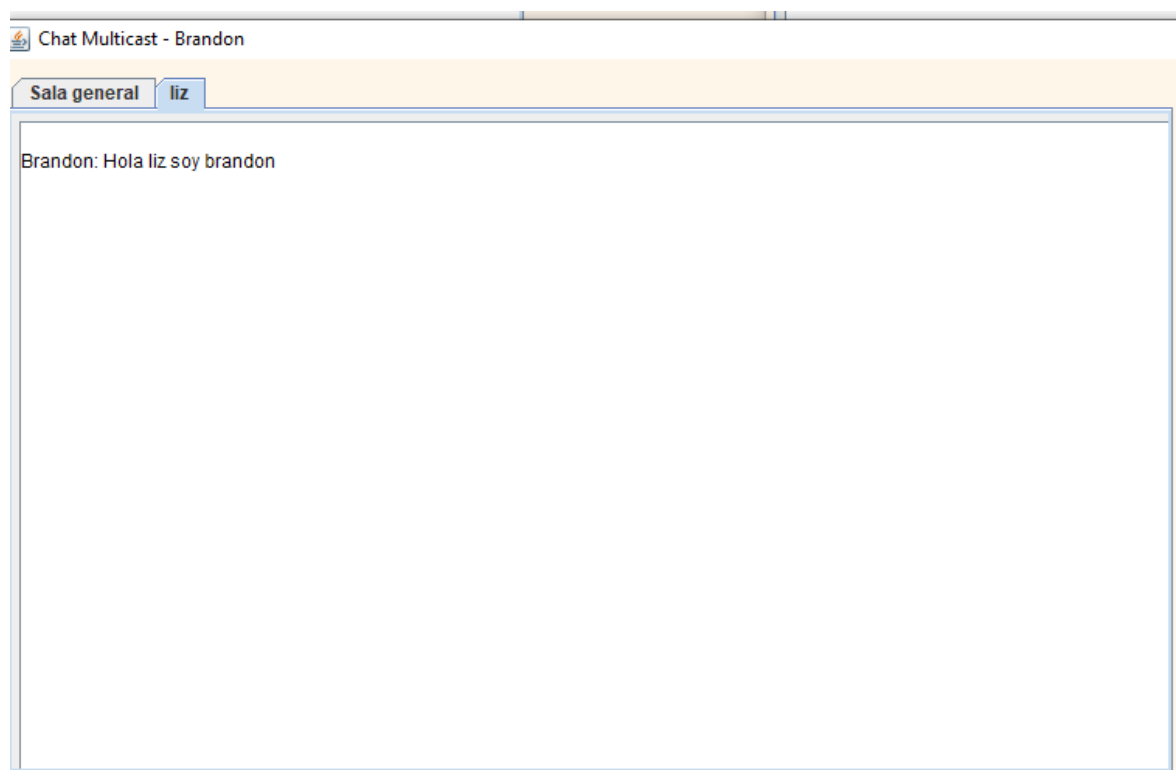
Ilustración 26. Envío de audio.

Para reproducir el audio se deberá pulsar en el botón que dice reproducir audio y se mostrará una lista de audios disponibles para reproducir.



*Ilustración 27. Reproduciendo audio,*

Para entablar una conversación privada se deberá pulsar en uno de los usuarios conectados mostrados en la parte de la derecha y se abrirá una nueva pestaña.



*Ilustración 28. Mensaje privado de brandon a liz*

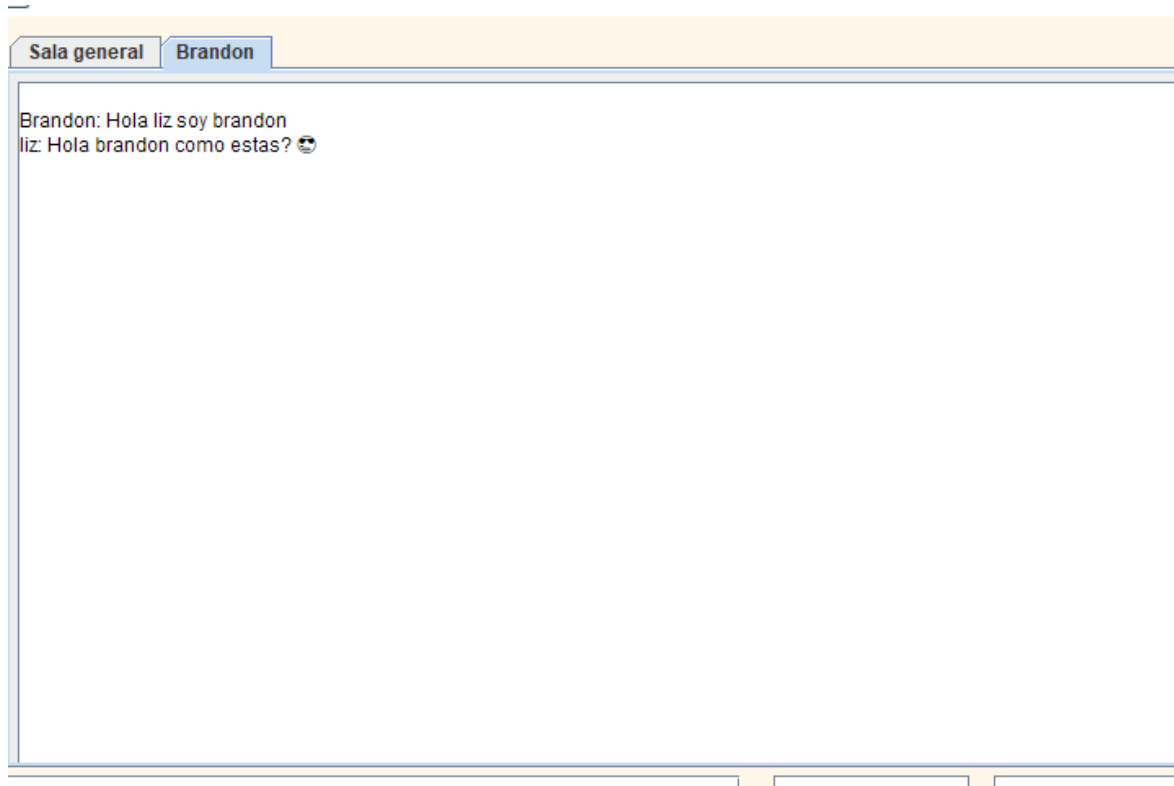


Ilustración 29. Mensaje de liz a brandon

En la parte inferior derecha se pueden encontrar emojis que los usuarios pueden mandar.



Ilustración 30. Mandando emojis.

Los caracteres en el chat se reemplazarán por los emojis una vez sean mandados como se ve a continuación.

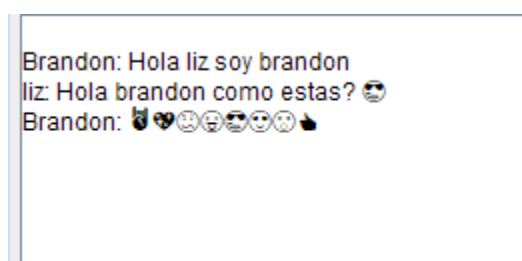


Ilustración 31. Emojis en el chat.

## **Conclusiones**

Para concluir, esta práctica me pareció una muy buena práctica para aplicar lo visto en clases sobre multicast, pues a partir de estos se logró realizar un chat con múltiples usuarios, cosa que se sigue viendo hoy en día con distintas aplicaciones entre las más famosas Facebook.

Durante el desarrollo de la práctica se me presentaron algunas complicaciones, entre las más destacables fue el hacer que se identifique el mensaje general o el mensaje privado, pero esto se resolvió gracias a la estructura de mensajes que nos recomendó el profesor, por medio de etiquetas se indicó si era un mensaje privado, el remitente y el receptor y cortando el mensaje y guardando esos cortes en nuevas variables se identificó si el mensaje era privado o general.

Sin duda una práctica retadora pero que al final se logró resolver de buena forma, una de las ventajas principales al usar sockets de datagrama o de multidifusión es que la información llega de manera rápida y segura.

## **Bibliografía**

1. programmer (2020). "Sockets de datagrama". Recuperado de :  
<https://programmerclick.com/article/638561752/>