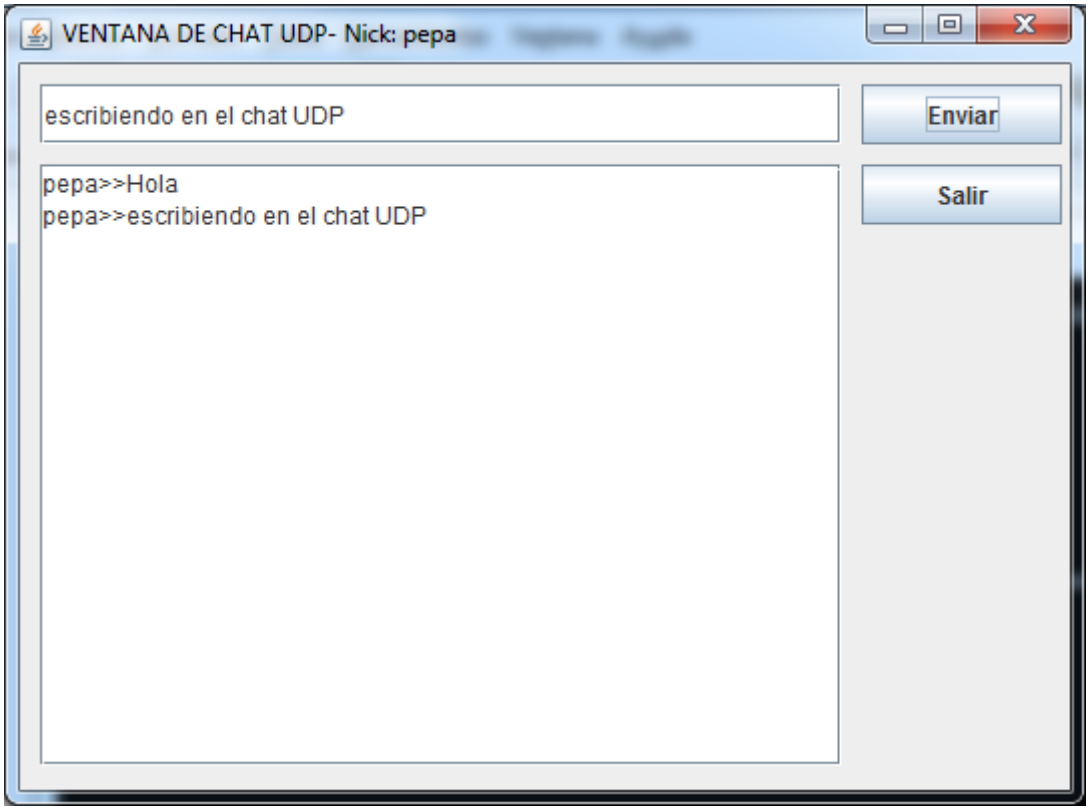
PROGRAMACIÓN DE COMUNICACIONES EN RED.

CHAT UDP.

Índice de contenido

1. CHAT UDP

A continuación vamos a crear un chat más sencillo utilizando **MulticastSocket**. Crearemos una única clase, *MultiChatUDP* que extiende **Runnable**, en la que se define una pantalla similar a la del cliente chat TCP. Tenemos 2 botones, uno para enviar el mensaje tecleado, otro para finalizar y un textarea donde se muestran los mensajes.



En el método *main()* se pide un nombre al usuario (nick), se crea un socket multicast en un puerto determinado, se configura la IP del grupo al que nos conectaremos, nos unimos al grupo para enviar y recibir mensajes, se comprueba si se ha escrito algo en el nombre, se muestra la pantalla y por último se lanza el hilo multichat:

```
public static void main(String args[]) throws IOException {
    String nombre = JOptionPane.showInputDialog("Introduce tu nombre o nick:");
    // Se crea el socket multicast
    ms = new MulticastSocket(Puerto);
    grupo = InetAddress.getByName("225.0.0.1");// Grupo multicast
    //Direcciones multicast entre 224.0.0.0 y 239.255.255.255
    // Nos unimos al grupo
    ms.joinGroup(grupo);
    if (!nombre.trim().equals("")) {
        MultiChatUDP server = new MultiChatUDP(nombre);
        server.setBounds(0, 0, 540, 400);
        server.setVisible(true) ;
        new Thread(server).start() ;//lanzar hilo
    }else {
    System. out. println ("El nombre está vacío .... ") ;
    }
}// main
```

Cada vez que se pulse el botón *Enviar* se envían los mensajes al grupo de multicast:

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == boton) { // SE PULSA ENVIAR
        String texto = nombre +">>" + mensaje.getText();
        try {
            // ENVIANDO mensaje al grupo
            DatagramPacket paquete = new DatagramPacket(texto.getBytes(),texto.length(), grupo, Puerto);
            ms.send(paquete);
        } catch (IOException el) { el.printStackTrace();}
    }//fin enviar
```

El botón *Salir* envia el mensaje de despedida al grupo y cierra el socket:

```
    if (e.getSource() == desconectar) { // SE PULSA SALIR
        String texto = "*** Abandona el chat: " + nombre + " ***";
        try {
            // ENVIANDO DESPEDIDA AL GRUPO
            DatagramPacket paquete = new DatagramPacket(texto.getBytes(), texto.length(), grupo, Puerto);
            ms.send(paquete) ;
            ms.close() ;
            repetir = false;
            System.out.println("Abandona el chat: "+ nombre);
            System.exit(0);
        }catch (IOException el) {el.printStackTrace();}
    }
}//actionPerformed
```

En el método *run()* del hilo se realiza un proceso repetitivo donde se muestran los mensajes que se reciben del grupo multicast en el textarea:

```
public void run() {
    while (repetir) {
        try {
            DatagramPacket p = new DatagramPacket(buf, buf.length);
            ms.receive(p);//recibo mensajes
            String texto = new String(p.getData(), 0, p.getLength());
            textarea1.append(texto + "\n");
        }catch (SocketException s) {System.out.println(s.getMessage());}
        catch (IOException e) {e.printStackTrace();}
    }
}// run
```

Por último se muestra la definición de las variables puerto, multicast y grupo, y la cabecera de la clase:

```
public class MultiChatUDP extends JFrame implements ActionListener, Runnable
    static MulticastSocket ms = null;
    static byte[] buf = new byte[1000];
    static InetAddress grupo = null;
    static int Puerto = 12345;// Puerto multicast
```

El resto de variables de pantalla son similares al ejemplo con TCP. Para probarlo se ejecuta el programa *MultiChatUDP* en las máquinas que quieran participar en el chat.

Basándonos en el código aquí mostrado y en el del ejemplo de chat TCP, es posible completar el MultiChatUDP.