UNIVERSIDAD NACIONAL AUTONOMA DE NICARAGUA
UNAN-LEON
FACULTAD DE CIENCIAS
DEPARTAMENTO DE COMPUTACION

CHAT INTERACTIVO DESARROLLADO EN JAVA PARA EL DEPARTAMENTO DE COMPUTACIÒN

MONOGRAFIA

Integrantes:

Br. Javier Enrique Hernández Sandoval
Br. Jairo Isaac Herrera Mora
Br. Edwin Alberto Penado Rodríguez

Para optar al titulo de:

LICENCIADO EN COMPUTACION

Tutor:

Msc. Álvaro Altamirano

León, Nicaragua, Mayo de 2005

INDICE

DEDICATORIA

Dedico esta monografía:

A DIOS: Padre Todopoderoso que día a día nos ilumina y nos guía por verdadero camino del bien.

A MI MADRE: Urania del Carmen Rodríguez que con su apoyo y verdadera abnegación me proporciono los medios necesario y de esta forma lograr culminar mi carrera.

A MI PADRE: Ronald Penado que me apoyo en todos estos los momentos de mi vida universitaria.

A MIS AMIGOS: Que me apoyaron en los buenos y malos momentos en el transcurso de mi carrera.

A MI NOVIA Y SU FAMILIA: Milena Hernández que me dio todo el apoyo emocional en mi estadía en la ciudad de León y su familia por su aceptación y apoyo moral.

A TODOS LOS DOCENTES: Que nos impartieron los conocimientos a lo largo de la carrera.

A TODOS ELLOS MUCHAS GRACIAS.

Edwin Alberto Penado Rodríguez.

DEDICATORIA.

Le dedico esta monografía a:

A MI MADRE: **Maria del Socorro Mora Rodríguez**

Inspiración de mis afanes, mujer luchadora abnegada del ideal su fe y las esperanzas en mi para obtener uno de mis grandes sueños **"ser un profesional"** quien con su sacrificio interés y apoyo en estos últimos años supo ayudarme incondicionalmente para coronar mi carrera.

A MI ESPOSA: **Marbell Mercedes Áreas.**

Razón de mi existencia y mis sueños, mujer comprensiva y cariñosa que con su ternura y amor y su gran apoyo en todos estos últimos años estuvo con migo en esos momentos difíciles y buenos.

A MIS HIJOS: **Jairo Isaac Herrera Areas**
Angie Junieth Herrera Areas

Flor de luz y de nuevos esplendores, mis grandes adoraciones y tesoros de mi corazón.

A MIS HERMANOS: Mi sangre, que estuvieron de acuerdo y apoyarme de una u otra forma en triunfar en mi carrera.

A MIS AMIGOS: Que de alguna u otra forma colaboraron con migo a través de sus consejos y por esos grandes momentos que pasamos juntos.

GRACIAS A TODOS ELLOS.

Jairo Isaac Herrera Mora.

DEDICATORIA

Dedico con gran orgullo esta monografía a:

A DIOS: El todo poderoso que me supo iluminar y guiar día a día por el buen camino de la vida y me dio sabiduría para lograr alcanzar mi meta mas deseada que es llegar a ser un profesional con valores éticos y humanísticos.

A MIS PADRES: **Msc. Máximo Román Hernández Cano**
Lic. Rosa Maria Sandoval Balladares

Quienes me han brindado su amor y apoyo incondicional y fueron parte fundamental en mi formación, por que gracias a su entrega y sacrificio pude llegar a culminar mis estudios.

A MIS ABUELOS: **Juan Sandoval Parajón (q.e.p.d)**
Isolina Balladares Gutiérrez

Porque los admiro y respeto y gracias a sus consejos he sabido caminar por el buen sendero.

A MIS HERMANOS: **Lic. Cristian R. Hernández S.**
Br. Mario A. Hernández S.

Que por su compañía y apoyo en momentos importantes de mi vida.

A MI SOBRINA: **Silvana Leonora Hernández U.**

Quien me ha regalado con su sonrisa alegría.

A MI NOVIA: **Maria Virginia Somarriba U.**

Por su amor, cariño y comprensión en los momentos difíciles de mi vida

A TODOS ELLOS GRACIAS.

JAVIER ENRIQUE HERNANDEZ SANDOVAL.

AGRADECIMIENTO

Queremos dar las gracias a nuestro padre celestial creador del cielo y la tierra, por habernos iluminado nuestras mentes y corazones para la elaboración de este trabajo y por darnos entendimiento en estos últimos cinco años de nuestra carrera y por culminarla de una manera realizada y satisfactoria.

A nuestro tutor Lic. Álvaro Altamirano por su apoyo en cada etapa de nuestro trabajo, su dedicación y habernos brindado sus conocimientos.

A los profesores que con sus conocimientos brindados hoy no hubiéramos hecho una realidad este trabajo y por habernos formado como profesionales del futuro.

Quisiéramos también expresar nuestro agradecimiento a las muchas personas que nos han prestado su ayuda en la elaboración de nuestro trabajo. Sus muestras de apoyos sus criticas y sugerencias han tenido para nosotros un valor inestimable.

A todos ellos GRACIAS.

I. INTRODUCCIÓN

La tecnología en el mundo de las comunicaciones no solamente ha revolucionado a grandes empresas, sino también a pequeñas y medianas, tanto que ha alcanzado un gran impacto a nivel mundial en su evolución.

Sabemos que su utilización es algo muy importante en distintas área de trabajo (Universidades, Colegios, CyberCafes, etc.). En base a lo anterior, nos hemos propuesto desarrollar una aplicación de Mensajeria (Chat) que sirva como un medio fácil y accesible para la transmisión y envío de datos en dichas áreas de trabajo.

II. ANTECEDENTES

Los inicios del IRC se remontan a 1988, cuando un finlandés llamado Jarkko Oikarinen escribió el código original. Fue por tanto en Finlandia donde se comenzó a usar esta tecnología, aunque en ese momento todavía no estaba en Internet, sino que J. Oikarinen la diseñó para usarla en su propia BBS como un sistema multichat en tiempo real.

Cuando comenzó a usar Internet como medio, el sistema comenzó a popularizarse rápidamente y pasó a convertirse en una herramienta de comunicación casi indispensable para todos aquellos que necesitaban comunicarse de una manera más directa que con el correo electrónico.

Hay dos fechas clave que marcaron el impulso definitivo del IRC. La primera es 1991, con el estallido de La Guerra del Golfo; el uso de este sistema de comunicación que plasmaba la realidad segundo a segundo comenzó a tomarse en serio. Fue en este momento cuando comenzaron a florecer los programas de IRC.

La otra fecha es Septiembre de 1993, cuando gran número de usuarios (en tiempo real) informaban desde Moscú de la inestabilidad social y política por la que estaba pasando el país.

Actualmente, los canales de conversación del IRC abarcan todos los temas imaginables, pudiendo encontrar canales en los que se habla de los temas más simples, hasta canales en donde los temas de conversación son absolutamente serios y de gran acervo cultural.

Hoy en día las redes más grandes de servidores de IRC son: Efnet, DALnet, UnderNet, NEWnet y Galaxynet.

En el contexto del Departamento de Computación vemos que no se ha implementado ningún mecanismo o software, que sirva de comunicación e intercambio de información a través de una red.

III. JUSTIFICACIÓN.

Comunicarse, para el hombre, es tan natural e imprescindible como respirar. Así en la actualidad vemos diferentes formas de hacerlo a través de redes LAN, Internet, etc., que sirven de gran ayuda en la comunicación de los usuarios.

Hoy en día en el Gremio del Departamento de Computación vemos que no se ha implementado ningún mecanismo o software que sirva de intercambio de datos a través de una red. Es por ello que desarrollamos una aplicación de Mensajeria (Chat) que sirva de comunicación en una red local (LAN) entre diferentes usuarios, la cual facilite el manejo de la información.

IV. OBJETIVO GENERAL

Establecer los lineamientos para la realización de un proyecto de un Chat interactivo desarrollado en el lenguaje java, que sirva de comunicación entre los usuarios que desean comunicarse, a través de una red.

OBJETIVOS ESPECÍFICOS

- ✓ Diseñar una interfaz amigable para el usuario parecida a la mayoría de los Chat actuales.

- ✓ Hacer uso del intercambio de mensajes entre los usuarios que están comunicados a través del Chat.

- ✓ Poder visualizar pequeñas imágenes en línea, enviadas por la persona con quien se conversa.

V. MARCO TEÓRICO

¿QUE ES UN CHAT?

Un Chat o IRC es un sistema de conversación multi-usuario, donde la gente se encuentra en "canales" (salas o emplazamientos virtuales, comúnmente con un tema de conversación) para hablar en grupos, o privadamente. No hay restricción en el número de gente que puede participar en una discusión determinada, o el número de canales que pueden formarse sobre IRC.

ELEMENTOS DE UN CHAT

Dentro de los elementos que encontramos dentro de un Chat para que se pueda llevar a cabo la comunicación, están los siguientes

Usuarios. Serán las personas que harán uso del Chat.

Canales. Donde los usuarios podrán entrar y salir, aunque en algunas se deban cumplir ciertos requisitos.

Chat Room Salas de Charla. Donde todos los usuarios "hablan" entre ellos

OPERS. Donde el/los usuario/s solicitan canales o cualquier tipo de información.

ADM (Administradores). Estos son los que marcan las pautas y normas a seguir para el buen funcionamiento del Chat y la conducta de los usuarios.

IrCOP. Serán las personas que se dedican al mantenimiento del Chat

OPER. Son las personas que ante las necesidades de los usuarios, les ayudan o suministran cualquier tipo de información respecto, comunicaciones entre canales, entre usuarios, reservas de canales privados, etc.

CARACTERÍSTICAS DE LOS CHAT´S

La tecnología de la CMC hace posible que un grupo de personas distantes físicamente, sin la posibilidad de verse el uno al otro puedan comunicarse de manera sincrónica, al igual que en los encuentros cara a cara, usando la palabra escrita. En esta forma de comunicación se combinan la permanencia de la palabra escrita y la fluidez del intercambio propia de las conversaciones presénciales.

Dentro de las características principales podemos mencionar:

Abierto las 24 horas del día todos los días. Internet y la totalidad de sus aplicaciones están disponibles las 24 horas del día todos los días. Sólo un par de clicks separan a la persona del acceso al mundo virtual si cuenta con el software y el hardware necesarios. Una vez ingresado (conectado) a la red, siempre habrá personas esperando alguien con quien conversar. Puede plantearse la posibilidad de que la persona frecuente un mismo Chat Room y que en éste, a las 7 de la mañana, no haya usuarios. Este pequeño problema se soluciona fácilmente: se puede entrar a otros canales de otros países (por ejemplo, al de España, que remite a un lugar del mundo donde son las 11 de la mañana y probablemente haya más usuarios en línea).

Control sobre la presentación de uno mismo y sobre lo que los otros ven del sí mismo. En IRC, el anonimato, facilita la creación de un personaje. Las máscaras esconden a la persona y permiten jugar un personaje cuyas características son fácilmente configuradas por la propia persona.

Control sobre la relación. Los programas de IRC ofrecen la posibilidad de elegir con quien hablar y con quien no. Es decir, que si al sujeto no le interesa comunicarse con una determinada persona, con sólo tipear un comando (/ignore) seguido por, por ejemplo, el nickname de ésta, logra su objetivo.

EL CHAT TIPO TEXTO

1. Permite la comunicación de dos o más personas por medio de texto
2. Permite crear grupos de dialogo o salones en los cuales se pueden compartir opiniones con los demás integrantes.

CHAT INTERACTIVO POR MEDIO DE VOZ

Permite comunicación por medio de voz

Esta comunicación puede ser de dos tipos:

1. Full dúplex (comunicación en doble vía, ejemplo una comunicación telefónica normal)
2. Half dúplex (comunicación en una sola vía, ejemplo la comunicación que se realiza con radio teléfono, se tiene que dar cambio no se puede hablar simultáneamente).
3. Provee bajos costos o costos locales en llamadas internacionales.

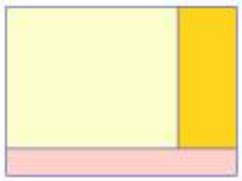
Con lo adelantada que esta la tecnología ya podemos encontrar Chat en 3D, algunos acompañados de videoconferencia. Dentro de algunos años quien sabe que más cosas le añadirán a los Chat.

FUNCIONAMIENTO DE UN CHAT

Un Chat es como una sala de reuniones donde quien entra se encuentra con un montón de gente con la que puede compartir texto, voz, video o programas desde tu ordenador. La sala es un servidor donde tu envías lo que quieres transmitir y él se encarga de reenviarlo en tiempo real a los restantes Chateros o a uno de ellos en concreto (en el caso de que el Chat admita privados).

Las personas que participan en la conversación usualmente eligen un nickname para evitar usar su nombre real y poder hablar con más libertad.

ESTRUCTURA BÁSICA DE UN CHAT



1. En la zona color crema se ven los mensajes de los participantes (también los tuyos).
2. En la zona color amarilla están listados por Nicks (apodos) los que participan.
3. En la zona color rosita ingresas tu texto, para que sea visualizado tienes que presionar Enter. Puede que haya un botón enviar.

Si observas con atención el Chat donde te encuentras tienes que descubrir en algún lugar un botón que diga privado, eso significa que seleccionando a un usuario (de la zona color amarilla) y presionando ese botón tendrás una conversación privada con el mismo. Puede ser que tengas que oprimir el botón derecho del Mouse sobre el Nick del usuario y ahí elegir esta alternativa. Otra opción muy frecuente es la de ignorar, para enmudecer a la persona que te está molestando (la manera es similar a la descripta en la opción privado).

TIPOS DE CHAT

¿QUÉ TIPOS DE CHAT EXISTEN?

Los hay de todo tipo, desde el que solo admite texto sobre un fondo liso hasta el que combina también voz e imagen junto con la posibilidad de compartir archivos, dibujar en una misma pizarra, etc. Claro esta que todos ellos se dividen o clasifican en dos clases: uno a través de una página Web y otro por IRC.

CLASES DE CHAT:

1. **Internet Relay Chat (IRC):** es un sistema común de chatear en el Internet. Solo accesas a un salón o canal en el IRC, en donde cada canal se enfoca a un tópico específico, tal como música o política. Para participar de estos grupos de dialogo necesitas tener un programa IRC como el que se encuentra en http://www.mirc.co.uk/

2. **Web-Based Chat:** Existen lugares en la red que te permiten hablar con otras personas y lo único que necesitas para participar es tu Web browser.

3. **Instant messaging:** Te permite chatear en privado con alguna persona en el Internet. Solo necesitas un programa especial como el que se encuentra en http://www.icq.com/

4. **Multimedia Chat:** Te permite tener una conversación hablada y una comunicación en video a través del Internet. Claro esta que necesitas un equipo especial para llevar a cabo esta actividad (bocinas, cámara de video, micrófono, etc.).

EVOLUCION DE LOS CHATS

MENSAJEROS INSTANTÁNEOS

La Red de Redes ha aportado al mundo de la telecomunicaciones grandes avances en la parcela de comunicación entre usuarios de Internet. Primero los correos electrónicos, después llegó el Chat y los últimos en sumarse a esta tendencia fueron los mensajeros instantáneos, una manera rápida de comunicarse con los demás.

El 15 de noviembre de 1996, cuatro meses después de su creación, la compañía Mirabilis Inc., compuesta por los israelíes Yair Goldfinger, Arik Vardi, Sefi Vigiser y Amnon Amir, presentó un software que revolucionó las comunicaciones en Internet. Medio año después, tenía un millón de usuarios. Y ya más de 127 millones de personas se registraron para usar el servicio. Estamos hablando, claro, del ICQ (derivado de la frase inglesa "I see you, te veo"), el programa que definió el concepto de mensajería instantánea, mezcla de Chat y e-mail que da la posibilidad de comunicarse en forma inmediata con otros usuarios de la Red en cualquier parte del

mundo. Permite saber cuál de sus amigos está conectado a la Web en cada momento, conectarse con él e iniciar una conversación tipo Chat, con respuesta instantánea.

Pero estos mensajeros permiten hacer más cosas que intercambiar textos. Las nuevas versiones pueden transmitir audio, video, datos, enviar mensajes a celulares, componer listas de tareas, tener páginas Web gratuitas, revisar el e-mail, etcétera. Y gratis: ninguno de los mensajeros instantáneos pide dinero para funcionar.

El último ICQ, por ejemplo, ofrece la opción de compartir archivos que estén en una carpeta designada (pero limitando el acceso al que el usuario quiera).

En mayo de 1997, AOL lanzó su propio mensajero. Y a mediados de 1998, la empresa compró a Mirabilis en 287 millones de dólares, poniendo en evidencia el valor del servicio. La acción fue seguida por los lanzamientos de Yahoo! en marzo de 1998 y Microsoft en julio del año siguiente.

Justamente, el MSN Messenger de la empresa de la ventanita tiene cada vez más seguidores, debido a su asociación al servicio de correo gratuito Hotmail, a la inclusión del software en el nuevo Windows XP (preinstalado con el nombre de Windows Messenger), y el peso de Microsoft como marca.

Merece ser probado, porque tiene buenas herramientas y es muy simple de usar. Los otros mensajeros instantáneos también tienen ventajas sobre el ICQ, así que puede buscar el que más se ajuste a sus necesidades.

CARACTERÍSTICAS DE LOS MENSAJEROS INSTANTÁNEOS

Son dinámicos y trabajan en tiempo real. Son cien por ciento interactivos. Sirven para contactar personas de cualquier punto del globo. Son el paradigma de la instantaneidad de la Red. Todos estos argumentos servirían para promocionar las nuevas aplicaciones estrella del entorno Web: los mensajeros instantáneos. Eso, en caso que éstos necesitaran promoción. Las cifras indican que ya han ganado buena fama sin mucho esfuerzo publicitario, porque llevan al extremo las ventajas de interactividad y velocidad de comunicación que la Red propone a través de sus múltiples servicios: según la consultora Mobile Insights, la mensajería instantánea en línea contará con 175 millones de usuarios en el 2002, lo que marca un incremento del 250 por ciento respecto de los 50 millones de cibernautas que actualmente cuentan con al menos uno de estos productos. Conocidos por las siglas PIM, Personal Instant Messengers, los programas de intercambio de mensajes en tiempo real funcionan como radares de la Red, que detectan a los amigos y conocidos listados en una suerte de libreta de direcciones, propia de estas aplicaciones y personalizada para cada usuario, y permiten establecer contacto con ellos.

EJEMPLOS DE MENSAJEROS INSTANTÁNEOS

Las características de los mensajeros depende de la compañía que los crea, así tenemos a:

ICQ

Lo bueno: Pionero en este tipo de programas cuenta con muchas funciones como compartir archivos de una de tus carpetas con otros usuario, cambiar sonidos, buscar amigos o mandar mensajes a móviles. Lo mejor de todo es mantener una conversación nueve personas a la vez en el mismo canal.

Lo malo: al principio cuesta acostumbrarse a su complicado interfaz. Además, si queremos conectarnos al ICQ desde otro ordenador, éste no nos guarda los contactos y deberemos introducirlos uno a uno, pero esta desventaja ya se ha corregido en la versión 2001.

AOL Instant Messenger

Lo bueno: Compartir todo tipo de archivos con tus amigos. Te avisa cuando alguien de tu lista de contactos se conecta. Poder mantener conversaciones de voz con los demás.

Lo malo: no es muy fácil de usar. Está en inglés.

Yahoo! Messenger

Lo bueno: Poder cambiar de pieles y personalizarlo tu gusto. La posibilidad de intercambiar archivos con tus amigos y te avisa de cuales de ellos se conectan. Hablar con los demás por Internet como si fuera un teléfono. Fácil de usar.

Lo malo: que no haya conseguido el éxito que se esperaba.

Instanterra

Lo bueno: al igual la gran mayoría de sus competidores te permite ver quienes de tus contactos están conectados.

Lo malo: se trata de un programa de Terra y poca gente de habla no hispana utiliza este servicio.

MSN Messenger

Lo bueno: enviar y recibir archivos de tus amistades. Mantener una conversación hablada, la versión que incluye Windows XP ofrece la posibilidad de videoconferencia. Permite hacer una lista negra de amigos ante los que uno puede aparecer invisible.

Una lista de tus contactos que te indica quienes de ellos permanecen conectados. Su interfaz sencillo y fácil de usar lo convierten en el más manejable de todos. Pero lo más destacado, poder conectarse desde cualquier ordenador que tenga instalado el Messenger.

Lo malo: no permite más de cuatro usuarios en la misma sala de Chat.

mIRC

Lo bueno: ofrece una multitud de canales. Mandar archivos a los demás. Los juegos tan divertidos a los que se pueden jugar como por ejemplo el ahorcado y el trivial.

Lo malo: que no tiene un sistema de correo electrónico. La larga lista de comandos para realizar distintas acciones.

Netmeeting

Lo bueno: permiten realizar llamadas mediante servidores de directorio, servidores de conferencia y páginas Web. NetMeeting permite que resulte más fácil realizar llamadas a través de Internet, la intranet de una organización, e incluso de un teléfono. los usuarios pueden enviar y recibir archivos para trabajar en ellos. Las características de vídeo y audio de NetMeeting permiten ver y oír a otras personas. Incluso si no puede transmitir vídeo, podrá recibir llamadas de vídeo en la ventana de vídeo de NetMeeting. Con la característica Conversación puede hablar con varias personas. Además, es posible codificar las llamadas de Conversación, lo que garantiza la privacidad de las conferencias. Mediante la Pizarra, puede dibujar la información para explicar conceptos, utilizar un esbozo o mostrar gráficos. También puede copiar áreas del escritorio o de las ventanas y pegarlas en la Pizarra.

Lo malo: tiene problemas de conexión.

MSN MESSENGER VS. OTROS CHAT´S

La versión 4.6 del MSN Messenger (en español, está en messenger.msn.es) es una buena alternativa al ICQ. Como su predecesor, permite enviar mensajes de texto (con tipos de letras, emoticones y colores) de manera instantánea entre los usuarios.

Para poder hacer uso de él, hay que tener una cuenta de Passport (http://www.passport.com), el servicio de identificación en la Web de Microsoft. Con tener una dirección de Hotmail es suficiente. Así como los usuarios de ICQ tienen un número único de identificación, el Messenger requiere una dirección de Hotmail para localizar a sus usuarios.

La interfaz es, quizás, el mayor logro del Messenger sobre el ICQ. Los botones son grandes y su uso es muy intuitivo y claro (agregar un usuario a la lista de amigos, iniciar una conversación, transferir un archivo, enviar un e-mail). Incluso ofrecen acciones fuera de lo que se supondría que atañe a una conversación virtual (como usar un juego de Microsoft instalado en la PC, etcétera). Una ventanita en el ángulo inferior derecho del monitor aparece cada vez que un amigo está en línea, llega un mensaje a la cuenta de Hotmail o alguien inicia una conversación.

Aunque es una herramienta útil (y más visible que el cambio del icono de la barra de tareas que suelen ofrecer los demás mensajeros), puede tornarse molesta en una sesión de Chat ajetreada. Para desactivarla, vaya a HerramientasOpciones> Preferencias>Avisos y destilde la acción no deseada.

También es posible iniciar una llamada a un teléfono convencional desde la PC y, en la versión incluida en el XP, hacer videoconferencia desde el mismo mensajero. Los demás deberán usar el NetMeeting. Y tiene una función excelente, hasta ahora exclusiva del Yahoo! Messenger: cuando el interlocutor está escribiendo un mensaje (y todavía no lo envió) aparece un cartelito en la ventana informando esto. Así sabemos que el otro está escribiendo una respuesta larga, y no que se fue a tomar un café y su regreso es incierto.

El Messenger permite hacer una lista negra de amigos ante los que uno puede aparecer invisible (en Herramientas>Opciones>Privacidad), pero lamentablemente no es posible personificar el mensaje en cada caso.

Si quiere mandar mensajes a todos sus amigos de una sola vez (una invitación a un cumpleaños, por ejemplo, o un anuncio formal), puede usar un plug-in como el Mass Messenger 1.2 (en http://madmaxnet.iespana.es/madmaxnet/tools/massmsg.htm), parecido a la función Recipientes múltiples del ICQ. El MSN Messenger es capaz de recibir alertas de noticias; sólo hay que seleccionar alguno de los botones de la barra vertical de anuncios de la izquierda.

Lo que le falta es poder configurar los comandos con atajos de teclado, como en el ICQ. Tiene algunos (Mayúsculas + Enter para crear un párrafo nuevo en una conversación sin enviar el texto, por ejemplo), pero son un poco limitados, y están fijos. A propósito, los atajos de teclado en el ICQ se cambian en MainPreferences> Contact List>Shortcuts.

LOS OTROS

ICQ y MSN Messenger son los más populares, pero no los únicos. Yahoo! Messenger (messenger.yahoo.com/intl/ar/) también es una buena alternativa.

A las herramientas básicas de todos los mensajeros (Chat, transferencia de archivos, etcétera) le agrega pieles para cambiar su apariencia. Y tiene una herramienta de videoconferencia integrada, muy cómoda para verse con otros usuarios del servicio. Además, permite armar una sesión de Chat con 9 usuarios, contra los 4 del MSN

Messenger. Lo único que se requiere para usar el servicio es una cuenta de correo de Yahoo! Tampoco hay que olvidar al AIM (http://www.aol.com.ar/Aim/aim40.adp), el popular mensajero de AOL, que a las herramientas usuales les agrega el poder definir un icono o imagen para cada amigo en la lista del mensajero (y ayudar a reconocer a cada uno).

Ahora bien: ¿qué sucede si un usuario de ICQ quiere comunicarse con otro de Yahoo! o Microsoft? Mientras la gente del Internet Engineering Task Force (IETF), la institución que se ocupa de elaborar algunos de los estándares que hacen funcionar a la Internet, termina de desarrollar un protocolo común para la mensajería instantánea (más datos en http://www.imppwg.org), las diferentes redes son incompatibles.

Pero hay varios servicios que actúan como puente entre unas y otras: el primero fue el del mensajero Odigo (http://www.odigo.com , en la instalación se puede elegir el idioma), que además de conectar a la gente de su propia red es capaz de comunicarse con las de ICQ, MSN, Yahoo! y AOL.

Así, si usted tiene amigos en esas redes, puede escribirles desde un solo lugar, sin andar abriendo y cerrando programas. Para comunicarse con cada uno, claro, tendrá que darse de alta primero en la red correspondiente.

Esto mismo ofrece Imici (http://www.imici.com) y Easy Message (http://www.ryandewsbury.com/easymessage/). Pero el campeón en interoperabilidad es el Trillian (http://www.trillian.cc).

CARAS DEL CHAT

El ambiente del Chat es bastante amigable con el uso del HTML y el Java.

CHAT BASADOS EN HTML

Se refiere a los usuarios conectados a un servidor de red corriendo propiamente una aplicación de Chat en un browser a través del HTTP, en donde los mensajes son enviados por la aplicación de Chat a cada cliente también vía HTTP.

CHAT BASADOS EN JAVA

Los Java client applet se comunican con el Java Chat Servlet a través de un servidor de red usando HTTP. Los Servlet procesan el mensaje, enviándolo devuelta al servidor de red para ser distribuido a cada uno de los clientes del Chat del Java vía HTTP.

SOCKETS

Son Mecanismos de comunicación entre programas a través de una red *TCP/IP*. Los procesos los tratan como descriptores de Archivos, de forma que se pueden intercambiar datos con otros procesos transmitiendo y recibiendo a través de sockets. Estos realizan la interfaz entre aplicación y protocolo. Dichos mecanismos pueden tener lugar dentro de la misma máquina o a través de una red, en forma cliente-servidor. Java proporciona para esto las clases ServerSocket, Socket, InetAddress, etc.

Para establecer la conexión a través de un *socket*, tenemos que programar por un lado el servidor y por otro los clientes. En el servidor utilizaremos la clase ServerSocket y en el cliente la clase Socket, utilizando el método accept() para esperar algún cliente. Una vez establecida la conexión podremos intercambiar datos utilizando los *Streams*.

SOCKETS STREAM (TCP, TRANSPORT CONTROL PROTOCOL)

Son un servicio orientado a conexión donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados.
El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets.

Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

SOCKETS DATAGRAMA (UDP, USER DATAGRAM PROTOCOL)

Son un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

SOCKETS RAW

Son sockets que dan acceso directo a la capa de software de red subyacente o a protocolos de más bajo nivel. Se utilizan sobre todo para la depuración del código de los protocolos.

DIFERENCIAS ENTRE SOCKETS STREAM Y DATAGRAMA

Ahora se nos presenta un problema, ¿qué protocolo, o tipo de sockets, debemos usar - UDP o TCP? La decisión depende de la aplicación cliente/servidor que estemos escribiendo. Vamos a ver algunas diferencias entre los protocolos para ayudar en la decisión.

En UDP, cada vez que se envía un datagrama, hay que enviar también el descriptor del socket local y la dirección del socket que va a recibir el datagrama, luego éstos son más grandes que los TCP. Como el protocolo TCP está orientado a conexión, tenemos que establecer esta conexión entre los dos sockets antes de nada, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no existe en UDP.

En UDP hay un límite de tamaño de los datagramas, establecido en 64 kilobytes, que se pueden enviar a una localización determinada, mientras que TCP no tiene límite; una vez que se ha establecido la conexión, el par de sockets funciona como los streams: todos los datos se leen inmediatamente, en el mismo orden en que se van recibiendo.

UDP es un protocolo *desordenado*, no garantiza que los datagramas que se hayan enviado sean recibidos en el mismo orden por el socket de recepción. Al contrario, TCP es un protocolo *ordenado*, garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado.

Los datagramas son bloques de información del tipo *lanzar y olvidar*. Para la mayoría de los programas que utilicen la red, el usar un flujo TCP en vez de un datagrama UDP es más sencillo y hay menos posibilidades de tener problemas. Sin embargo, cuando se requiere un rendimiento óptimo, y está justificado el tiempo adicional que supone realizar la verificación de los datos, los datagramas son un mecanismo realmente útil.

En resumen, TCP parece más indicado para la implementación de servicios de red como un control remoto (*rlogin*, *telnet*) y transmisión de Archivos (*ftp*); que necesitan transmitir datos de longitud indefinida. UDP es menos complejo y tiene una menor sobrecarga sobre la conexión; esto hace que sea el indicado en la implementación de aplicaciones cliente/servidor en sistemas distribuidos montados sobre redes de área local.

USO DE SOCKETS

Podemos pensar que un *Servidor Internet* es un conjunto de sockets que proporciona capacidades adicionales del sistema, los llamados *servicios*.

SERVIDORES

Como se ha visto antes Java admite conectores de servidor. Los ServerSocket se utilizan para crear servidores en Internet (programas que están esperando a programas clientes locales). Estos conectores tienen un método adicional accept() que se bloquea y espera que un cliente inicie la comunicación para devolver un Socket y devolver la comunicación al cliente.

OBJETO SERVERSOCKET:

ServerSocket obj_server=new ServerSocket (int puerto);

OBJETO SOCKET: para poder recibir al cliente.

Socket obj_cliente;

RECIBIR LA COMUNICACIÓN (ACCEPT):

Obj_cliente=obj_server.accept ();

DEVOLVER LA COMUNICACIÓN(PRINTSTREAM): muy parecido a un println.

```
PrintStream obj;
obj=new PrintStream(obj_cliente.getOutputStream());
```

CLIENTES

Los conectores TCP/IP (*Socket*) se utilizan para implementar conexiones entre nodos de Internet. Estos conectores pueden examinar en cualquier momento la información de dirección y puerto asociado con ellos y acceder a los flujos E/S que se han visto anteriormente.

CREAR SOCKET: dos métodos para la creación.

Socket obj_clie=new Socket (direccion, int puerto);

Socket obj_clie=new Socket (String nodo, int puerto);

METODOS ASOCIADOS:

obj_clie.*metodo*();

metodo()	***DESCRIPCIÓN***
getInetAddress()	Devuelve la dirección a la que esta conectado.
getPort()	Devuelve el puerto remoto que esta conectado.
getLocalPort()	Devuelve el puerto local que esta conectado.
getInputStream()	Devuelve el flujo de entrada al cliente.
getOutputStream()	Devuelve el flujo de salida al cliente.
close()	Cierra los Streams.

La clase **InetAddress** se utiliza para obtener los nombres de los nodos y sus direcciones IP. Los métodos de esta clase están siempre asociados a *Socket*. La excepción que lanzan en caso de error es UnknownHostException. Nos van a servir de gran apoyo a la hora de utilizar los Socket. La clase InetAddress no tiene constructores visibles, son métodos estáticos.

BUSCAR NODO LOCAL:

InetAddress obj=InetAddress.getLocalHost ();

BUSCAR NODO POR NOMBRE:
InetAddress obj=InetAddress.getByName ("nombre");

BUSCAR TODOS LOS NODOS POR NOMBRE:
InetAddress obj []=InetAddress.getAllByName ("nombre");

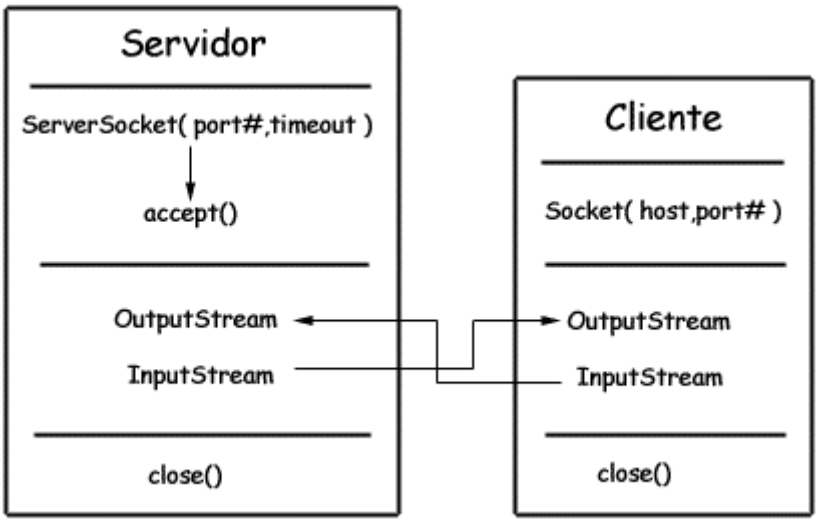
OBTENER NOMBRE DE HOST:
String=obj_socket.getHostName ();

OBTENER DIRECCION:
byte[]=obj_socket.getAddress();

OBTENER NOMBRE E IP DEL NODO:
String=obj_socket.toString ();

MODELO DE COMUNICACIONES CON JAVA

En Java, crear una conexión socket TCP/IP se realiza directamente con el paquete **java.net**. A continuación mostramos un diagrama de lo que ocurre en el lado del cliente y del servidor:



El modelo de Sockets más simple es:

1. El servidor establece un puerto y espera durante un cierto tiempo (**timeout** segundos), a que el cliente establezca la conexión. Cuando el cliente solicite una conexión, el servidor abrirá la conexión Socket con el método *accept()*.
2. El cliente establece una conexión con la máquina **host** a través del puerto que se designe en **puerto#.**
3. El cliente y el servidor se comunican con manejadores InputStream y OutputStream.

Hay una cuestión al respecto de los Sockets, que viene impuesta por la implementación del sistema de seguridad de Java. Actualmente, los applets sólo pueden establecer conexiones con el nodo desde el cual se transfirió su código. Esto está implementado en el JDK y en el intérprete de Java de Netscape. Esto reduce en gran manera la flexibilidad de las fuentes de datos disponibles para los applets. El problema si se permite que un applet se conecte a cualquier máquina de la red, es que entonces se podrían utilizar los applets para inundar la red desde un ordenador con un cliente Netscape del que no se sospecha y sin ninguna posibilidad de rastreo.

VI. PROCESO DE LA INSTALACIÓN

INSTALACIÓN DE LA APLICACIÓN CLIENTE

1. Ejecutar el archivo ChatCompCliente 

2. Se mostrara el proceso de instalación y dar clic en el botón next

3. El programa será copiado en el destino que el usuario eligió y se creara un acceso directo en Grupo de programas donde será ejecutado, el usuario podrá crear un acceso directo en el escritorio si lo desea.

INSTALACIÓN DE LA APLICACIÓN SERVIDOR

1. Ejecutar el archivo ChatCompServidor 

2. Se mostrara el proceso de instalación y dar clic en el botón next

3. El programa será copiado en el destino que el usuario eligió y se creara un acceso directo en Grupo de programas donde será ejecutado, el usuario podrá crear un acceso directo en el escritorio si lo desea.

REQUISITOS PARA LA INSTALACION DE LA APLICACIÓN:

1. Microsoft Window 95/98/2000/Me/XP
2. Maquina virtual de java (jre1.3 o superior)

VII. METODOLOGÍA DEL TRABAJO

Para la elaboración de esta aplicación (Chat Interactivo), el tipo de Sistema Operativo que usaremos será Windows. Esta podrá ser implementada bajo plataforma “Linux y Windows”, debido a que el tipo del lenguaje a usar será Java. La metodología a emplear para la elaboración de este trabajo monográfico será basada en el modelo de Ciclo de vida en cascada:

1. Análisis de la aplicación.

2. Diseño de la Aplicación:
   2.1. Cliente
   2.2. Servidor
3. Codificación de la Aplicación.

4. Prueba de la aplicación Cliente-Servidor
   4.1. Funcionamiento de la aplicación Cliente-Servidor
   4.2. Establecimiento de la comunicación Cliente-Servidor.

5. Instalación de la Aplicación Cliente-Servidor.

VIII. RECURSOS DISPONIBLES Y NECESARIOS

Recursos Hardware

1. Intel Celeron 950 MHz
2. Disco Duro de 60 Gb
3. Memoria 192 MB
4. Impresora hp

Recursos Software

1. Jdk 1.4.2 (Compilador de Java)
2. Microsoft Word
3. Microsoft PowerPoint

IX. FASE DE ANALISIS

ESPECIFICACIÓN DE REQUISITOS DEL CHAT

1. Introducción

1.1 Propósito

Definición del conjunto de especificaciones de los requisitos del software que debe cumplir la aplicación del Chat Interactivo desarrollado en el lenguaje java, que sirva de comunicación entre los usuarios que desean comunicarse, a través de una red TCP/IP.

Este documento será dirigido al departamento de Computación y los usuarios finales que deberán de estudiarlo para su aprobación o desacuerdo antes de abordar la fase de análisis.

1.2 Alcance

El nombre con el que se conocerá a esta aplicación será: ChatComp.

La Aplicación realizara las siguientes funciones:

1. Intercambio de texto plano entre los usuarios.

2. Intercambio de imágenes en la sala de Chat.

1.3 Definiciones, Acrónimos y Abreviaturas

Texto Plano: Texto escrito por el usuario a través del teclado, para luego ser enviado a otro usuario.
Imágenes: Pequeñas imágenes llamadas Emoticons en formato gif.
Apodo: Nombre con que se conocerá al usuario en la sala.

1.4 Referencias

Información obtenida a través de un Pequeño tutorial sobre una conexión cliente/servidor en java
http://www.geocities.com/chuidiang/java/sockets/socket.html

Pequeño Manual del uso de Sockets
http://www.cica.es/formacion/JavaTut/Cap9/sock_uso.html

Pagina que publica monografías de usuarios
http://www.monografias.com

JAVA El lenguaje de programación de Internet Tutoriales de JAVA extraídos de:
http://fibers.upc.es/pages/tutorials.php

1.5 Visión General

Primeramente se realizará una Descripción General de la Aplicación que se desea desarrollar para pasar posteriormente a estudiar cada uno de los Requisitos Específicos individualmente.

2. Descripción General

2.1 Relaciones con la aplicación

La Chat interactuará como una aplicación Cliente / Servidor a través de la red del Departamento de Computación.

El equipo en el que se desarrollara la aplicación cliente es:

1. PC compatible
2. 128 MB RAM
3. 500 MB de espacio
4. Maquina virtual de Java

El equipo en el que se desarrollara la aplicación servidor es:

1. PC compatible o Servidor
2. 256 RAM
3. 500 MB
4. Maquina Virtual de Java

La instalación inicial constara de 2 ordenadores uno cliente y otro servidor.

2.2 Funciones de la aplicación Cliente

La aplicación cliente debe contener todas las tareas que realiza el usuario al conectarse al servidor y poder conversar a través de texto e intercambiar imágenes con otros usuarios también conectados al servidor:

1. Conexión a la sala del Chat
2. Conversar a través de texto
3. Compartir pequeñas imágenes dentro de la sala
4. Desconexión de la sala del Chat.
5. Reconexión a la sala del Chat.

2.3 Funciones de la aplicación Servidor

La aplicación servidor debe contener todas las tareas necesarias para que el administrador pueda activar, desactivar el servidor y desconectar a cualquier usuario no deseado.

1. Activar el servidor.
2. Desconectar usuario de la sala.
3. Desactivar el servidor.

2.4 Características del usuario

Los usuarios finales del Chat serán personas que no se le exigen mucha experiencia en el área de la informática.

2.5 Restricciones Generales

El lenguaje de programación a utilizar será JAVA.

3. Requisitos Específicos.

3.1 Requisitos Funcionales de la aplicación Cliente

3.1.1 Conexión a la sala del Chat

3.1.1.1 Especificación

3.1.1.1.1 Introducción

Cuando el usuario desea ser miembro de la sala tendrá que llenar un determinado número de datos para su conexión.

3.1.1.1.2 Entradas

Por pantalla:

1. Apodo
2. Presionar botón de Conexión.
3. IP del Servidor.

3.1.1.1.3 Proceso

Se mostrara por pantalla una caja de dialogo pidiendo el apodo con el que se identificara en la sala y si es por primera vez que se ejecuta se le pedirá la dirección IP del servidor.

3.1.1.1.4 Salidas

Se mostrara el apodo en la lista de usuarios conectados en la sala del Chat.

3.1.1.2 Interfaces Externas

3.1.1.2.1 Interfaces de Usuarios

La captura del Apodo se realizara de forma interactiva.

3.1.1.2.2 Interfaces de Hardware

Se podrá utilizar cualquier computador conectado al servidor.

3.1.1.2.3 Interfaces Software

El proceso interactuara directamente con el Servidor.

3.1.1.2.4 Interfaces de Comunicación

Existe una interfaz de comunicación en la aplicación cliente para interactuar directamente con el servidor.

3.1.2 Conversar a través de texto

3.1.2.1 Especificación

3.1.2.1.1 Introducción

Establecida la conexión y estando en la sala del Chat, podrá compartir texto para el envío de mensajes por medio del teclado.

3.1.2.1.2 Entradas

Por pantalla:
Mensajes de texto por medio del teclado.

3.1.2.1.3 Proceso

Se mostrara por pantalla una caja de texto donde se podrá introducir los mensajes y podrán ser enviados y mostrados en la sala del Chat.

3.1.2.1.4 Salidas

Se mostrara el mensaje enviado en una caja de texto multilínea en sala del Chat.

3.1.2.2 Interfaces externas

3.1.2.2.1 Interfaces de usuario

La captura del texto se hará de forma interactiva.

3.1.2.2.2 Interfaces Hardware

Se podrá utilizar cualquier computador conectado al servidor.

3.1.2.2.3 Interfaces Software

El proceso interactuara directamente con el servidor.

3.1.2.2.4 Interfaces de Comunicación

Existe una interfaz de comunicación en la aplicación cliente para interactuar directamente con el servidor.

3.1.3 Compartir pequeñas imágenes dentro de la sala

3.1.3.1 Especificación

3.1.3.1.1 Introducción

Establecida la conexión, podrá compartir imágenes con los demás usuarios mostrándose la imagen en la misma sala de charla.

3.1.3.1.2 Entradas

Por pantalla:
Lista de imágenes.

3.1.3.1.3 Proceso

Se mostrara una lista de imágenes para que luego pueda ser seleccionada y enviada a los demás usuarios de la sala.

3.1.3.1.4 Salidas

Se mostrara la imagen dentro de la sala del Chat.

3.1.3.2 Interfaces externas

3.1.3.2.1 Interfaces de usuario

La captura de la imagen se hará de forma interactiva.

3.1.3.2.2 Interfaces Hardware

Se podrá utilizar cualquier computador conectado al servidor.

3.1.3.2.3 Interfaces Software

El proceso interactuara directamente con el servidor.

3.1.3.2.4 Interfaces de Comunicación

Existe una interfaz de comunicación en la aplicación cliente para interactuar directamente con el servidor.

3.1.4 Desconexión de la sala del Chat

3.1.4.1 Especificación.

3.1.4.1.1 Introducción

Cuando el usuario desea abandonar la sala, se tendrá que presionar el botón correspondiente para la desconexión.

3.1.4.1.2 Entradas

Por pantalla:
Presionar botón de desconexión.

3.1.4.1.3 Proceso

El usuario deberá desconectarse de la sala presionando el botón de desconexión.

3.1.4.1.4 Salidas

Mensaje mostrado en la caja de texto multilínea o sala de Chat que ha sido desconectado.

3.1.4.2 Interfaces externas

3.1.4.2.1 Interfaces de usuario

La desconexión se hará de forma interactiva.

3.1.4.2.2 Interfaces Hardware

Se podrá utilizar cualquier computador conectado al servidor.

3.1.4.2.3 Interfaces Software

El proceso interactuara directamente con el servidor.

3.1.4.2.4 Interfaces de Comunicación

Existe una interfaz de comunicación en la aplicación cliente para interactuar directamente con el servidor.

3.1.5 Reconexión a la sala del Chat

3.1.5.1 Especificación.

3.1.5.1.1 Introducción

El usuario tendrá la oportunidad de reconectarse a la sala después de haberla abandonado y cambiar su apodo con el que se identifica dentro de la sala.

3.1.5.1.2 Entradas

Por pantalla:
Se mostrara una caja de dialogo pidiendo su nuevo apodo

3.1.5.1.3 Proceso

Por pantalla se le pedirá que introduzca su nuevo apodo y así de nuevo volver a conectarse.

3.1.5.1.4 Salidas

Mostrara el nuevo apodo en la lista de usuarios conectados y en su conversación.

3.1.5.2 Interfaces externas

3.1.5.2.1 Interfaces de usuario

La reconexión a la sala se hará de forma interactiva.

3.1.5.2.2 Interfaces Hardware

Se podrá utilizar cualquier computador conectado al servidor.

3.1.5.2.3 Interfaces Software

El proceso interactuara directamente con el servidor.

3.1.5.2.4 Interfaces de Comunicación

Existe una interfaz de comunicación en la aplicación cliente para interactuar directamente con el servidor.

3.2 Requisitos funcionales de la aplicación Servidor.

3.2.1 Activar servidor

3.2.1.1 Especificación

3.2.1.1.1 Introducción

El administrador del Chat tendrá primero que activar el servidor para que los usuarios puedan conectarse

3.2.1.1.2 Entradas

Por pantalla:
Presionar botón Activar.

3.2.1.1.3 Proceso

Se tendrá que presionar el botón activar, para que los clientes puedan conectarse a el y el administrador pueda ver los usuarios conectados en una lista.

3.2.1.1.4 Salidas

Se mostrara en una lista el usuario que acaba de conectarse.

3.2.1.2 Interfaces Externas

3.2.1.2.1 Interfaces de Usuarios

El programa servidor interactuara directamente con el programa cliente.

3.2.1.2.2 Interfaces de Hardware

Se podrá utilizar cualquier Computador como Servidor en espera de que un cliente se conecte.

3.2.1.2.3 Interfaces Software

El proceso interactuara directamente con el programa cliente.

3.2.1.2.4 Interfaces de Comunicación

Existe una interfaz de comunicación en la aplicación servidor para interactuar directamente con el cliente.

3.2.2 Desconectar a un usuario

3.2.2.1 Especificación

3.2.2.1.1 Introducción

El administrador del Chat tendrá el privilegio de desconectar a usuarios.

3.2.2.1.2 Entradas

Por pantalla:
Presionar botón Desconectar.

3.2.2.1.3 Proceso

Por pantalla el administrador tendrá la opción de desconectar a cualquier usuario que esta siendo molesto.

3.2.2.1.4 Salidas

El usuario ya no aparecerá en la lista de usuarios conectados.

3.2.2.2 Interfaces Externas

3.2.2.2.1 Interfaces de Usuarios

El programa servidor interactuara directamente con el programa cliente.

3.2.2.2.2 Interfaces de Hardware

Se podrá utilizar cualquier Computador como Servidor en espera de que un cliente se conecte.

3.2.2.2.3 Interfaces Software

El proceso interactuara directamente con el programa cliente.

3.2.2.2.4 Interfaces de Comunicación

Existe una interfaz de comunicación en la aplicación servidor para interactuar directamente con el cliente.

3.2.3 Desactivar Servidor

3.2.3.1 Especificación

3.2.3.1.1 Introducción

El administrador del Chat tendrá la opción de ya no brindar el servicio y desactivar el servidor.

3.2.3.1.2 Entradas

Por pantalla:
Presionar botón Desactivar

3.2.3.1.3 Proceso

Por pantalla el administrador tendrá la opción de desactivar el servicio totalmente.

3.2.3.1.4 Salidas

Mensaje indicando que el servidor ha sido desactivado.

3.2.3.2 Interfaces Externas

3.2.3.2.1 Interfaces de Usuarios

El programa servidor interactuara directamente con el programa cliente.

3.2.3.2.2Interfaces de Hardware

Se podrá utilizar cualquier Computador como Servidor en espera de que un cliente se conecte.

3.2.3.2.3 Interfaces Software

El proceso interactuara directamente con el programa cliente.

3.2.3.2.4 Interfaces de Comunicación

Existe una interfaz de comunicación en la aplicación servidor para interactuar directamente con el cliente.

3.3 Requisitos de funcionamiento.

Requisitos estáticos: no existe ninguna restricción sobre el número de terminales o de usuarios que estén trabajando en este sistema.

3.4 Restricciones de diseño.

En pantalla se enlistara el apodo de cada usuario y en un área de texto los mensajes de cada uno de ellos.

3.5 Atributos.

Seguridad.

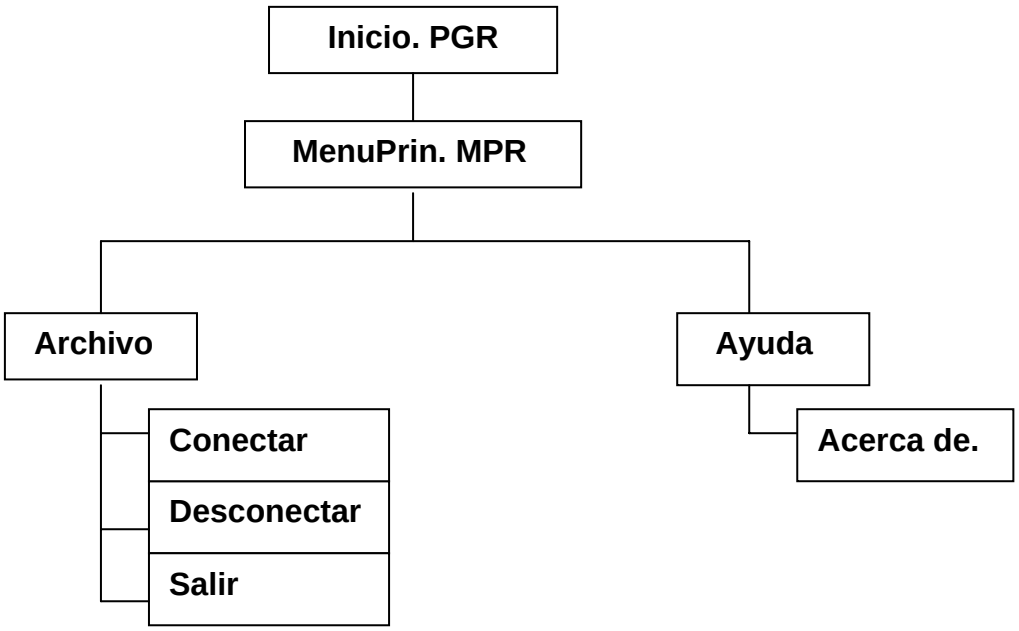
Tanto la aplicación Cliente como la del Servidor no brindaran ninguna seguridad en la transferencia de datos por parte de los usuarios conectados.

3.5.2 Mantenimiento.

Cualquier modificación que afecte a los requisitos mencionados en este documento deberá ser reflejada en el mismo, así como la documentación obtenida en las fases de análisis, diseño y programación.

X. FASE DE DISEÑO

DISEÑO ARQUITECTÓNICO DE LA APLICACIÓN CLIENTE



DISEÑO DE LA INTERFAZ

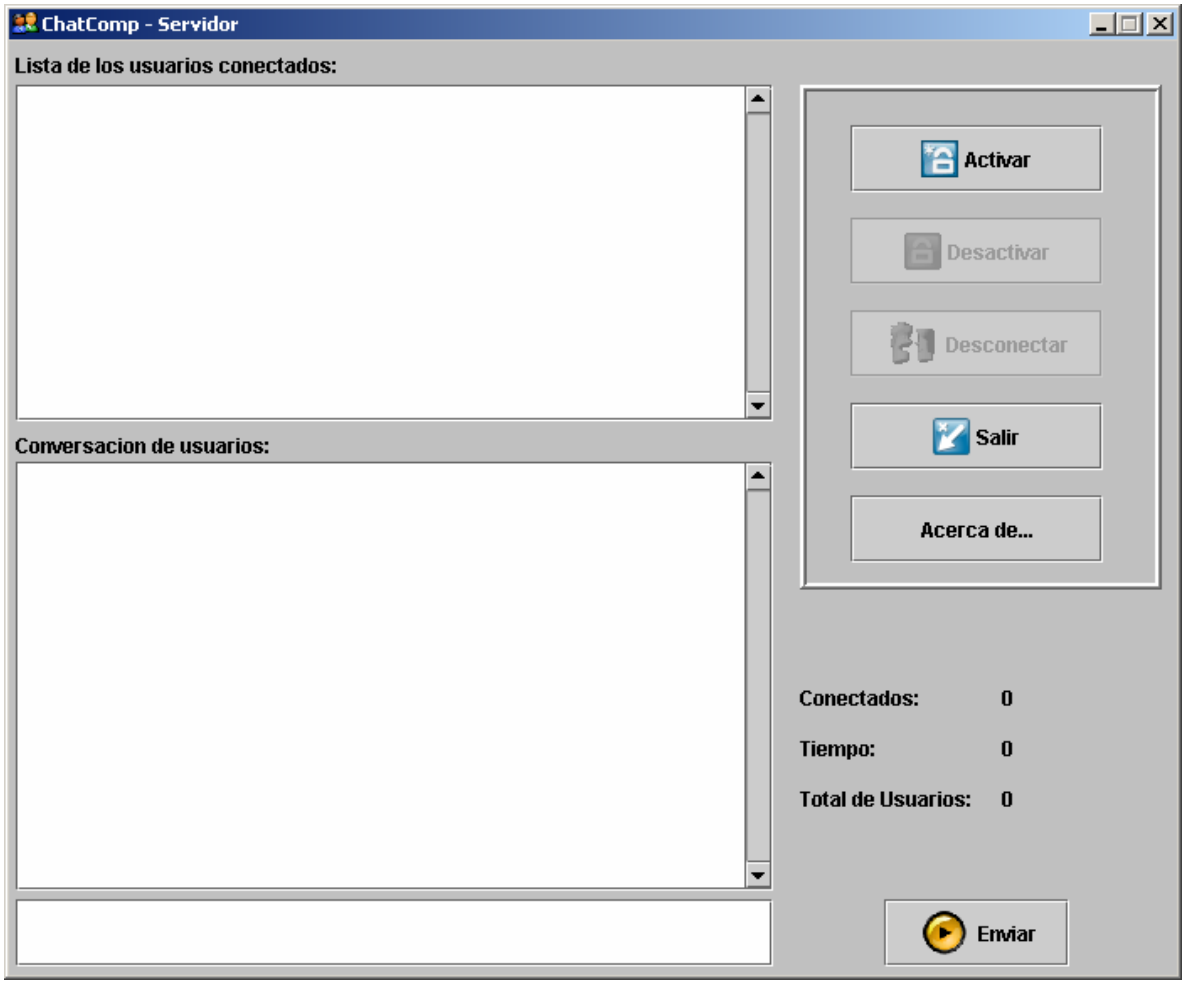
INTERFACES EXTERNAS

INTERFAZ DE USUARIO

PROGRAMA SERVIDOR

INTERFAZ PRINCIPAL DE LA APLICACIÓN SERVIDOR

El programa servidor contendrá la funcionalidad de activar, desactivar, desconectar un usuario conectado y salir de la aplicación.



Dialogo que muestra el botón **Acerca de.**

INTERFAZ DE USUARIO

PROGRAMA CLIENTE

La barra de herramienta tendrá las opciones de conexión y desconexión de la sala.

Menú Principal de la aplicación cliente.

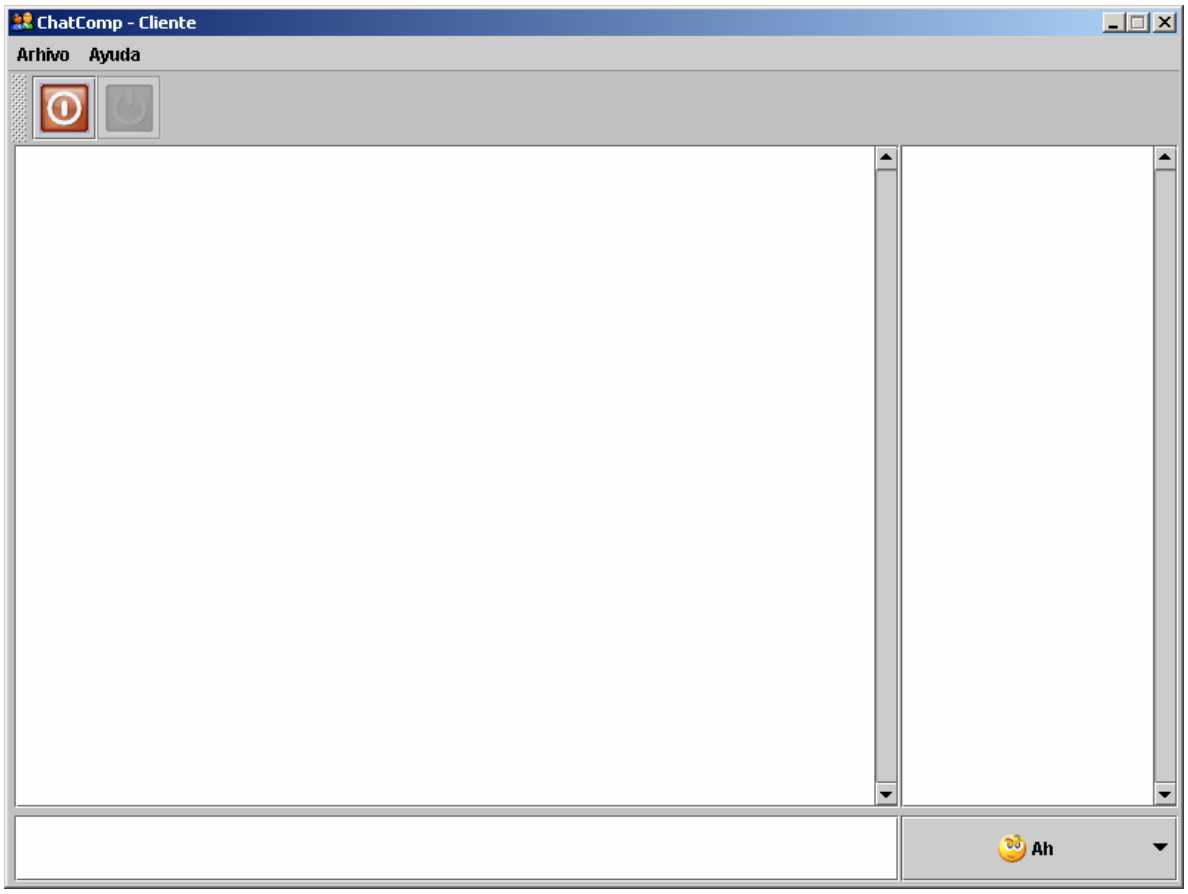
Este es el menú que aparecerá luego que se haya ejecutado la aplicación cliente del Chat.

La opción **Archivo** contendrá las opciones para que un usuario pueda conectarse, desconectarse o reconectarse a la sala.

La opción **Ayuda** solamente contendrá la opción de Acerca de nuestra aplicación.

La lista combinada contendrá todas las imágenes que el usuario puede enviar a la sala.

INTERFAZ PRINCIPAL DE LA APLICACIÓN CLIENTE



Dialogo que muestra la opción del menú ayuda **Acerca de.**

XI. CODIFICACIÓN.

La aplicación cliente esta constituida por los siguiente ficheros java:

ChatCliente.java fichero principal

Splash.java pantalla de inicio de la aplicación

Acercade.java ventana de dialogo acerca de

ChatCliente contiene todo el código esencial para la comunicación con el servidor, es el siguiente:

```
public void run()
{
   //El vector no tiene ninguna direccion ip
   this.ActualizarVector();
   if(ip.size() == 0)
   {
      String direcc = (String)JOptionPane.showInputDialog(this,"IP Servidor:");
      If(direcc != null && (direcc.length() > 0))
      {
         ip.addElement(direcc);
         this.EscribirDatos();
          apodo = (String)JOptionPane.showInputDialog(this,"Introduce tu Apodo:");
          String msg = "";
          //Se escribio algo!
          if ((apodo != null) && (apodo.length() > 0))
          {
             Desconectar.setEnabled(true);
             Conectar.setEnabled(false);
             OpcionesMenuArchivoConectar.setEnabled(false);
             OpcionesMenuArchivoDesconectar.setEnabled(true);
             conectar();
             Envio(apodo);
             while (true)
             {
                 try
                 {
                    try
                    {
                        msg = (String)i.readObject();// Se lee el objeto
                     }catch(ClassNotFoundException classNot){}
                    if(msg.equals("Ya existe Apodo"))
                    {
                       JOptionPane.showMessageDialog(this,"Escoja otro
apodo","Advertencia!",JOptionPane.INFORMATION_MESSAGE);
                       Conectar.setEnabled(false);
                       Conectar.setVisible(false);
                       OpcionesMenuArchivoConectar.setEnabled(false);
                       OpcionesMenuArchivoConectar.setVisible(false);
                       Reconectar.setEnabled(true);
                       Reconectar.setVisible(true);
                       OpcionesMenuArchivoReconectar.setEnabled(true);
                       OpcionesMenuArchivoReconectar.setVisible(true);
```

```
Desconectar.setEnabled(false);
OpcionesMenuArchivoDesconectar.setEnabled(false);
i.close();
o.close();
socket_cliente.close();
msg = "/ping";
}
else if(msg.equals("Has sido eliminado por el administrador"))
{
     Conectar.setEnabled(false);
     Conectar.setVisible(false);
OpcionesMenuArchivoConectar.setEnabled(false);
    OpcionesMenuArchivoConectar.setVisible(false);
     Reconectar.setEnabled(true);
     Reconectar.setVisible(true);
OpcionesMenuArchivoReconectar.setEnabled(true);
OpcionesMenuArchivoReconectar.setVisible(true);
    Desconectar.setEnabled(false);
OpcionesMenuArchivoDesconectar.setEnabled(false);
     conectado = false;
     lista.removeAllElements();
     i.close();
     o.close();
     msg = "/ping";
}
else if(msg.equals("/Imagen"))
{
  try
  {
      try
      {
        String ap = (String)i.readObject();
        String imag = (String)i.readObject();
        ImageIcon icono = new ImageIcon("Emoticons/" + imag + ".png");
        StyleConstants.setIcon(s, icono);
        doc.insertString(doc.getLength(),ap + ": ",doc.getStyle("large"));
        doc.insertString(doc.getLength()," " + "\n",doc.getStyle("icon"));
      }catch(ClassNotFoundException classNot){}
    } catch (BadLocationException ble){}
   msg = "/ping";
 }
 else if(msg.equals("/Lista"))
 {
    try
   {
      lista = (Vector)i.readObject();// Se lee el objeto
      usuarios_conectados.setListData(lista);
   }catch(ClassNotFoundException classNot){}
   msg = "/ping";
 }
 else if(msg.equals("/logout"))
 {
     try
     {
       String usu = (String)i.readObject();
       String aux;
```

```
int s = 0;
while(lista.size() > s)
{
    aux = (String)lista.get(s);
    if(aux.equals(usu))
    {
       lista.remove(s);
    }
    s++;
}
usuarios_conectados.setListData(lista);
 try
 {
   doc.insertString(doc.getLength(),usu + " ha salido de la sala" + "\n",doc.getStyle("large"));
 }catch(BadLocationException ble) {}
}catch(ClassNotFoundException classNot){}
msg = "/ping";
}
if(!msg.equals("/ping"))
{
   try
   {
     doc.insertString(doc.getLength(),msg + "\n",doc.getStyle("large"));
    }catch(BadLocationException ble) {}
}
}catch (IOException e)
{
    try
    {
      Thread.sleep(1010);
    }catch (InterruptedException e1)
    {
      e1.printStackTrace();
    }
}//Cierre de catch
}//Cierre del while
}//Cierre de if
else //no se escribio nada
{
   Desconectar.setEnabled(false);
   Conectar.setEnabled(true);
   OpcionesMenuArchivoConectar.setEnabled(true);
   OpcionesMenuArchivoDesconectar.setEnabled(false);
}
}//Abre if
else//no se escribio nada
{
    Desconectar.setEnabled(false);
    Conectar.setEnabled(true);
    OpcionesMenuArchivoConectar.setEnabled(true);
    OpcionesMenuArchivoDesconectar.setEnabled(false);
}
}//Abre if
else
{
   apodo = (String)JOptionPane.showInputDialog(this,"Introduce tu Apodo:");
   String msg = "";
```

```
    //Se escribio algo!
    if ((apodo != null) && (apodo.length() > 0))
    {
        Desconectar.setEnabled(true);
        Conectar.setEnabled(false);
        OpcionesMenuArchivoConectar.setEnabled(false);
        OpcionesMenuArchivoDesconectar.setEnabled(true);
        conectar();
        Envio(apodo);
        while (true)
        {
           try
           {
              try
              {
                  msg = (String)i.readObject();// Se lee el objeto
               }catch(ClassNotFoundException classNot){}
              if(msg.equals("Ya existe Apodo"))
              {
JOptionPane.showMessageDialog(this,"Escoja otro
apodo","Advertencia!",JOptionPane.INFORMATION_MESSAGE);
                  Conectar.setEnabled(false);
                  Conectar.setVisible(false);
                  OpcionesMenuArchivoConectar.setEnabled(false);
                  OpcionesMenuArchivoConectar.setVisible(false);
                  Reconectar.setEnabled(true);
                  Reconectar.setVisible(true);
OpcionesMenuArchivoReconectar.setEnabled(true);
                 OpcionesMenuArchivoReconectar.setVisible(true);
                 Desconectar.setEnabled(false);
OpcionesMenuArchivoDesconectar.setEnabled(false);
i.close();
o.close();
socket_cliente.close();
msg = "/ping";
}
else if(msg.equals("Has sido eliminado por el administrador"))
{
    Conectar.setEnabled(false);
    Conectar.setVisible(false);
OpcionesMenuArchivoConectar.setEnabled(false);
OpcionesMenuArchivoConectar.setVisible(false);
Reconectar.setEnabled(true);
     Reconectar.setVisible(true);
    OpcionesMenuArchivoReconectar.setEnabled(true);
OpcionesMenuArchivoReconectar.setVisible(true);
    Desconectar.setEnabled(false);
OpcionesMenuArchivoDesconectar.setEnabled(false);
    conectado = false;
    i.close();
    o.close();
    lista.removeAllElements();
usuarios_conectados.setListData(lista);
try
{
    doc.insertString(doc.getLength(),msg + "\n",doc.getStyle("large"));
}catch(BadLocationException ble) {}
```

```
msg = "/ping";
}
else if(msg.equals("/Imagen"))
{
  try
  {
     try
     {
        String ap = (String)i.readObject();
        String imag = (String)i.readObject();
        Style s = doc.addStyle("icon", def);
        StyleConstants.setAlignment(s, StyleConstants.ALIGN_CENTER);
        ImageIcon icono = new ImageIcon("Emoticons/" + imag + ".png");
        StyleConstants.setIcon(s, icono);
        doc.insertString(doc.getLength(),ap + ": ",doc.getStyle("large"));
        doc.insertString(doc.getLength()," " + "\n",doc.getStyle("icon"));
      }catch(ClassNotFoundException classNot){}
    } catch (BadLocationException ble){}
    msg = "/ping";
 }
else if(msg.equals("/Lista"))
{
  try
  {
      lista = (Vector)i.readObject();// Se lee el objeto
      usuarios_conectados.setListData(lista)
}catch(ClassNotFoundException classNot){}
   msg = "/ping";
}
else if(msg.equals("/logout"))
{
     try
    {
       String usu = (String)i.readObject();
       String aux;
       int s = 0;
       while(lista.size() > s)
       {
          aux = (String)lista.get(s);
          if(aux.equals(usu))
          {
             lista.remove(s);
          }
          s++;
        }
        usuarios_conectados.setListData(lista);
        try
        {
         doc.insertString(doc.getLength(),usu + " ha salido de la sala" +
"\n",doc.getStyle("large"));
         }catch(BadLocationException ble) {}
       }catch(ClassNotFoundException classNot){}
       msg = "/ping";
   }
```

```
if(!msg.equals("/ping"))
{
    try
    {
        doc.insertString(doc.getLength(),msg + "\n",doc.getStyle("large"));
    }catch(BadLocationException ble) {}
 }
}catch (IOException e)
{
    try
    {
       Thread.sleep(1010);
    }catch (InterruptedException e1)
    {
          e1.printStackTrace();
    }
 }//Cierre de catch
}//Cierre del while
}
else //no se escribio nada
{
    Desconectar.setEnabled(false);
    Conectar.setEnabled(true);
    OpcionesMenuArchivoConectar.setEnabled(true);
    OpcionesMenuArchivoDesconectar.setEnabled(false);
}
}//Cierre del else
}//Cierre del run
```

La aplicación Servidor esta constituida por los siguiente ficheros java:

ChatServidor.java fichero principal

Oyente.java Clase que acepta la petición de los clientes

MiniServidor.java hilo que se crea cuando un cliente se conecta

El código principal de la aplicación se encuentra distribuido en los ficheros Oyente.java y MiniServidor.java

Código principal del Oyente.java

```
public void run()
{
    try
    {
        miSocket = new ServerSocket(2004);
        anyadir("Servidor Conectado");
        conectado=true;
        while (true)
        {
            Socket cliente = miSocket.accept();
            //si estamos aki es que no ha habido un error
            anyadir("Conectandose un usuario: ");
            anyadir("   "+cliente.getInetAddress().toString());
            //tenemos que ponerlos en las labels

ChatServidor.Conectados2.setText(String.valueOf(Integer.parseInt(ChatServidor.Conectados2.g
etText())+1));
ChatServidor.Conectados_Inicio2.setText(String.valueOf(Integer.parseInt(ChatServidor.Conecta
dos_Inicio2.getText())+1));

            MiniServidor m = new MiniServidor(cliente,this);
          }
        }catch (IOException ioe)
        {
            conectado=false;
            //anyadir("Error "+ioe.toString());
        }
}
```

Código principal de MiniServidor.java

```
public void run()
{
  try
  {
        String msg;
        nombreUsuario = (String)i.readObject();// Se lee el objeto;
        if(hilos.size() >= 1)
        {
           int z = 0;
           //Verificacion de Duplicados...
           while(z < hilos.size())
           {
              respuesta = (MiniServidor) hilos.get(z);
              if(nombreUsuario.equals(respuesta.nombreUsuario))
              {
                doble = true;
                hilos.addElement(this);
                int x = 0;
                while(x < hilos.size())
                {
                    MiniServidor duplicado = (MiniServidor)hilos.get(x);
                    int tamano = hilos.size();
                    if(x == tamano - 1)
                    {
                       duplicado.o.writeObject("Ya existe Apodo");
                    }
                    duplicado.o.flush();
                    x++;
                 }
                 //Eliminamos el duplicado
                 hilos.remove(this);
                 break;
              }
              else
              {
                 doble = false;
              }
              respuesta.o.flush();
              z++;
         }
       }
       if(doble == false)
       {
           hilos.addElement(this);
           vectorListaUsuarios.addElement(nombreUsuario);
           ChatServidor.usuarios_conectados.setListData(vectorListaUsuarios);
           EscribirDatos();
           //Enviamos la orden "/Lista"
           int k =0;
           String lis = "/Lista";
```

```
while(k < hilos.size())
{
   respuesta= (MiniServidor) hilos.get(k);
   respuesta.o.writeObject(lis);
   respuesta.o.flush();
   ActualizarVector();
   respuesta.o.writeObject(this.vectorListaUsuarios);
   respuesta.o.flush();
   k++;
}
Oyente.anyadir("   "+String.valueOf(sckt.hashCode())+" / nick= "+nombreUsuario);
//Les escribimos a todos los usuarios quien acaba de iniciar sesion
int g = 0;
while(hilos.size() > g)
{
   respuesta= (MiniServidor) hilos.get(g);
   respuesta.o.writeObject(nombreUsuario + " ha entrado en la sala");
   respuesta.o.flush();
   g++;
}
while (true)
{
     msg = (String)i.readObject();// Se lee el objeto
     if(msg.equals("/Imagen"))
     {
        String apodo = (String)i.readObject();
        String imagen = (String)i.readObject();
        String image = "/Imagen";
        int j=0;
        while(hilos.size() > j)
        {
           respuesta= (MiniServidor) hilos.get(j);
           respuesta.o.writeObject(image);
           respuesta.o.writeObject(apodo);
           respuesta.o.writeObject(imagen);
           respuesta.o.flush();
           j++;
        }
        msg = " ";
     }
     else if(msg.equals("/logout"))
     {
         int a = 0;
         String  salida = "/logout";
         while(hilos.size() > a)
         {
           respuesta= (MiniServidor) hilos.get(a);
           respuesta.o.writeObject(salida);
           respuesta.o.writeObject(nombreUsuario);
           respuesta.o.flush();
           a++;
         }
         //el usuario ha pedido la desconexion y ha de ser dado de baja
         ChatServidor.Conectados2.setText(String.valueOf(Integer.parseInt(ChatServidor.Conect
ados2.getText())-1));
         Oyente.anyadir("El usuario "+nombreUsuario+" ha sido eliminado");
```

```
            MiniServidor.eliminaUsuario(nombreUsuario,false);
        }
        else if(!msg.equals(" "))
        {
            Oyente.anyadir(nombreUsuario+": "+msg);
         }
         int j=0;
         while(hilos.size() > j)
        {
            respuesta= (MiniServidor) hilos.get(j);
            if(msg.equals(" "))
            {
                break;
            }
            respuesta.o.writeObject(nombreUsuario + ": " + msg);
            respuesta.o.flush();
            j++;
        }//Fin del while
    }//Fin del While infinito
}//Fin del boolean
}catch (IOException ioe)
{}
catch (Throwable t)
{}
}//Cierre de run
```

XII. CONCLUSIÓN.

Después de haber analizado las características propias del lenguaje Java, podemos concluir que hasta el momento, es un software de programación en el que podemos confiar al diseñar nuestras aplicaciones, esto es, por su gran versatilidad, facilidad de programación y seguridad.

Los Chat´s en nuestra época están dando un gran giro comercial, ya que no solo se utiliza para intercomunicar a un grupo de gente, sino que también, para establecer una relación entre los usuarios.

Al concluir nuestro trabajo monográfico hemos logrados cumplir con nuestros objetivos planteados, hacer el trabajo de comunicación mucho mas fácil.

Por tanto al termino de este trabajo hemos adquiridos una mayor comprensión de las técnicas utilizadas para la realización del mismo.

XIII. RECOMENDACIÓN.

Una vez en uso el Chat y aprobado por las personas que lo utilizaran, se podrá mejorar implementando otras funcionalidades, como la transmisión de ficheros, uso de salas privadas entre los usuarios, etc.

Se recomienda que al servidor se le de un mantenimiento diario al menos un par de horas para lograr que todo marche bien.

XIV. BIBLIOGRAFÍA

- Pequeño tutorial sobre una conexión cliente/servidor en java
  http://www.geocities.com/chuidiang/java/sockets/socket.html

- Pequeño Manual del uso de Sockets
  http://www.cica.es/formacion/JavaTut/Cap9/sock_uso.html

- Pagina que publica monografías de usuarios
  http://www.monografias.com

- JAVA El lenguaje de programación de Internet Tutoriales de JAVA extraídos de:
  http://fibers.upc.es/pages/tutorials.php

Anexos

java.net
Class Socket

java.lang.Object
  └ **java.net.Socket**
Direct Known Subclasses:
SSLSocket

---

public class **Socket**
extends Object

This class implements client sockets (also called just "sockets"). A socket is an endpoint for communication between two machines.

The actual work of the socket is performed by an instance of the SocketImpl class. An application, by changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall.

Since:
JDK1.0
See Also:
setSocketImplFactory(java.net.SocketImplFactory), SocketImpl, SocketChannel

---

Constructor Summary	
	Socket() Creates an unconnected socket, with the system-default type of SocketImpl.
	Socket(InetAddress address, int port) Creates a stream socket and connects it to the specified port number at the specified IP address.
	Socket(InetAddress host, int port, boolean stream) **Deprecated.** *Use DatagramSocket instead for UDP transport.*
	Socket(InetAddress address, int port, InetAddress localAddr, int localPort) Creates a socket and connects it to the specified remote address on the specified remote port.
protected	**Socket**(SocketImpl impl) Creates an unconnected Socket with a user-specified SocketImpl.
	Socket(String host, int port) Creates a stream socket and connects it to the specified port number on the named host.
	Socket(String host, int port, boolean stream) **Deprecated.** *Use DatagramSocket instead for UDP transport.*

Socket(String host, int port, InetAddress localAddr, int localPort) Creates a socket and connects it to the specified remote host on the specified remote port.

Method Summary	
void	**bind**(SocketAddress bindpoint) Binds the socket to a local address.
void	**close**() Closes this socket.
void	**connect**(SocketAddress endpoint) Connects this socket to the server.
void	**connect**(SocketAddress endpoint, int timeout) Connects this socket to the server with a specified timeout value.
SocketChannel	**getChannel**() Returns the unique SocketChannel object associated with this socket, if any.
InetAddress	**getInetAddress**() Returns the address to which the socket is connected.
InputStream	**getInputStream**() Returns an input stream for this socket.
boolean	**getKeepAlive**() Tests if SO_KEEPALIVE is enabled.
InetAddress	**getLocalAddress**() Gets the local address to which the socket is bound.
int	**getLocalPort**() Returns the local port to which this socket is bound.
SocketAddress	**getLocalSocketAddress**() Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
boolean	**getOOBInline**() Tests if OOBINLINE is enabled.
OutputStream	**getOutputStream**() Returns an output stream for this socket.
int	**getPort**() Returns the remote port to which this socket is connected.
int	**getReceiveBufferSize**() Gets the value of the SO_RCVBUF option for this Socket, that is the buffer size used by the platform for input on this Socket.
SocketAddress	**getRemoteSocketAddress**() Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
boolean	**getReuseAddress**() Tests if SO_REUSEADDR is enabled.

boolean	**getReuseAddress**() Tests if SO_REUSEADDR is enabled.
int	**getSendBufferSize**() Get value of the SO_SNDBUF option for this Socket, that is the buffer size used by the platform for output on this Socket.
int	**getSoLinger**() Returns setting for SO_LINGER.
int	**getSoTimeout**() Returns setting for SO_TIMEOUT.
boolean	**getTcpNoDelay**() Tests if TCP_NODELAY is enabled.
int	**getTrafficClass**() Gets traffic class or type-of-service in the IP header for packets sent from this Socket
boolean	**isBound**() Returns the binding state of the socket.
boolean	**isClosed**() Returns the closed state of the socket.
boolean	**isConnected**() Returns the connection state of the socket.
boolean	**isInputShutdown**() Returns wether the read-half of the socket connection is closed.
boolean	**isOutputShutdown**() Returns wether the write-half of the socket connection is closed.
void	**sendUrgentData**(int data) Send one byte of urgent data on the socket.
void	**setKeepAlive**(boolean on) Enable/disable SO_KEEPALIVE.
void	**setOOBInline**(boolean on) Enable/disable OOBINLINE (receipt of TCP urgent data) By default, this option is disabled and TCP urgent data received on a socket is silently discarded.
void	**setReceiveBufferSize**(int size) Sets the SO_RCVBUF option to the specified value for this Socket.
void	**setReuseAddress**(boolean on) Enable/disable the SO_REUSEADDR socket option.
void	**setSendBufferSize**(int size) Sets the SO_SNDBUF option to the specified value for this Socket.
static void	**setSocketImplFactory**(SocketImplFactory fac) Sets the client socket implementation factory for the application.
void	**setSoLinger**(boolean on, int linger) Enable/disable SO_LINGER with the specified linger time in seconds.

void	**setSoTimeout**(int timeout) Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
void	**setTcpNoDelay**(boolean on) Enable/disable TCP_NODELAY (disable/enable Nagle's algorithm).
void	**setTrafficClass**(int tc) Sets traffic class or type-of-service octet in the IP header for packets sent from this Socket.
void	**shutdownInput**() Places the input stream for this socket at "end of stream".
void	**shutdownOutput**() Disables the output stream for this socket.
String	**toString**() Converts this socket to a String.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Socket

public **Socket**()

Creates an unconnected socket, with the system-default type of SocketImpl.

Since:

JDK1.1

Socket

protected **Socket**(SocketImpl impl)
throws SocketException

Creates an unconnected Socket with a user-specified SocketImpl.

Parameters:

impl - an instance of a **SocketImpl** the subclass wishes to use on the Socket.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

JDK1.1

Socket

public **Socket**(String host,
int port)
throws UnknownHostException,
IOException

Creates a stream socket and connects it to the specified port number on the named host.

If the specified host is null it is the equivalent of specifying the address as InetAddress.getByName(null). In other words, it is equivalent to specifying an address of the loopback interface.

If the application has specified a server socket factory, that factory's createSocketImpl method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException.

Parameters:

host - the host name, or null for the loopback address.
port - the port number.

Throws:

UnknownHostException - if the IP address of the host could not be determined.
IOException - if an I/O error occurs when creating the socket.
SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.

See Also:

setSocketImplFactory(java.net.SocketImplFactory), SocketImpl, SocketImplFactory.createSocketImpl(), SecurityManager.checkConnect(java.lang.String, int)

Socket

public **Socket**(InetAddress address,
int port)
throws IOException

Creates a stream socket and connects it to the specified port number at the specified IP address.

If the application has specified a socket factory, that factory's createSocketImpl method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException.

Parameters:

address - the IP address.
port - the port number.

Throws:

IOException - if an I/O error occurs when creating the socket.

SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.

See Also:

setSocketImplFactory(java.net.SocketImplFactory), SocketImpl, SocketImplFactory.createSocketImpl(), SecurityManager.checkConnect(java.lang.String, int)

Socket

public **Socket**(String host,
int port,
InetAddress localAddr,
int localPort)
throws IOException

Creates a socket and connects it to the specified remote host on the specified remote port. The Socket will also bind() to the local address and port supplied.

If the specified host is null it is the equivalent of specifying the address as InetAddress.getByName(null). In other words, it is equivalent to specifying an address of the loopback interface.

If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException.

Parameters:

host - the name of the remote host, or null for the loopback address.
port - the remote port
localAddr - the local address the socket is bound to
localPort - the local port the socket is bound to

Throws:

IOException - if an I/O error occurs when creating the socket.
SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.

Since:

JDK1.1

See Also:

SecurityManager.checkConnect(java.lang.String, int)

Socket

public **Socket**(InetAddress address,
int port,
InetAddress localAddr,
int localPort)
throws IOException

Creates a socket and connects it to the specified remote address on the specified remote port. The Socket will also bind() to the local address and port supplied.

If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException.

Parameters:

address - the remote address
port - the remote port
localAddr - the local address the socket is bound to
localPort - the local port the socket is bound to

Throws:

IOException - if an I/O error occurs when creating the socket.
SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.

Since:

JDK1.1

See Also:

SecurityManager.checkConnect(java.lang.String, int)

Socket

public **Socket**(String host,
int port,
boolean stream)
throws IOException

Deprecated. *Use DatagramSocket instead for UDP transport.*
Creates a stream socket and connects it to the specified port number on the named host.

If the specified host is null it is the equivalent of specifying the address as InetAddress.getByName(null). In other words, it is equivalent to specifying an address of the loopback interface.

If the stream argument is true, this creates a stream socket. If the stream argument is false, it creates a datagram socket.

If the application has specified a server socket factory, that factory's createSocketImpl method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException.

If a UDP socket is used, TCP/IP related socket options will not apply.

Parameters:

host - the host name, or null for the loopback address.
port - the port number.
stream - a boolean indicating whether this is a stream socket or a datagram socket.

Throws:

IOException - if an I/O error occurs when creating the socket.
SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.

See Also:
setSocketImplFactory(java.net.SocketImplFactory), SocketImpl, SocketImplFactory.createSocketImpl(), SecurityManager.checkConnect(java.lang.String, int)

Socket

public **Socket**(InetAddress host,
int port,
boolean stream)
throws IOException

Deprecated. *Use DatagramSocket instead for UDP transport.*
Creates a socket and connects it to the specified port number at the specified IP address.

If the stream argument is true, this creates a stream socket. If the stream argument is false, it creates a datagram socket.

If the application has specified a server socket factory, that factory's createSocketImpl method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its checkConnect method is called with host.getHostAddress() and port as its arguments. This could result in a SecurityException.

If UDP socket is used, TCP/IP related socket options will not apply.

Parameters:
host - the IP address.
port - the port number.
stream - if true, create a stream socket; otherwise, create a datagram socket.

Throws:
IOException - if an I/O error occurs when creating the socket.
SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.

See Also:
setSocketImplFactory(java.net.SocketImplFactory), SocketImpl, SocketImplFactory.createSocketImpl(), SecurityManager.checkConnect(java.lang.String, int)

Method Detail

connect

public void **connect**(SocketAddress endpoint)
throws IOException

Connects this socket to the server.
Parameters:
endpoint - the SocketAddress
Throws:

IOException - if an error occurs during the connection
IllegalBlockingModeException - if this socket has an associated channel, and the channel is in non-blocking mode
IllegalArgumentException - if endpoint is null or is a SocketAddress subclass not supported by this socket
Since: 1.4

---

connect

public void **connect**(SocketAddress endpoint,
int timeout)
throws IOException

Connects this socket to the server with a specified timeout value. A timeout of zero is interpreted as an infinite timeout. The connection will then block until established or an error occurs.
Parameters:
endpoint - the SocketAddress
timeout - the timeout value to be used in milliseconds.
Throws:
IOException - if an error occurs during the connection
SocketTimeoutException - if timeout expires before connecting
IllegalBlockingModeException - if this socket has an associated channel, and the channel is in non-blocking mode
IllegalArgumentException - if endpoint is null or is a SocketAddress subclass not supported by this socket
Since: 1.4

---

bind

public void **bind**(SocketAddress bindpoint)
throws IOException

Binds the socket to a local address.

If the address is null, then the system will pick up an ephemeral port and a valid local address to bind the socket.

Parameters:
bindpoint - the SocketAddress to bind to
Throws:
IOException - if the bind operation fails, or if the socket is already bound.

IllegalArgumentException - if bindpoint is a SocketAddress subclass not supported by this socket
Since:
1.4
See Also:
isBound()

---

getInetAddress

public InetAddress **getInetAddress**()
Returns the address to which the socket is connected.
Returns:
the remote IP address to which this socket is connected, or null if the socket is not connected.

---

getLocalAddress

public InetAddress **getLocalAddress**()
Gets the local address to which the socket is bound.
Returns:
the local address to which the socket is bound or InetAddress.anyLocalAddress() if the socket is not bound yet.
Since: JDK1.1

---

getPort

public int **getPort**()
Returns the remote port to which this socket is connected.
Returns:
the remote port number to which this socket is connected, or 0 if the socket is not connected yet.

---

getLocalPort

public int **getLocalPort**()
Returns the local port to which this socket is bound.
Returns:
the local port number to which this socket is bound or -1 if the socket is not bound yet.

---

getRemoteSocketAddress

public SocketAddress **getRemoteSocketAddress**()
Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
Returns:
a SocketAddress reprensenting the remote endpoint of this socket, or null if it is not connected yet.
Since: 1.4
See Also:
getInetAddress(), getPort(), connect(SocketAddress, int), connect(SocketAddress)

getLocalSocketAddress

public SocketAddress **getLocalSocketAddress**()

Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
Returns:
a SocketAddress representing the local endpoint of this socket, or null if it is not bound yet.
Since: 1.4
See Also:
getLocalAddress(), getLocalPort(), bind(SocketAddress)

---

getChannel

public SocketChannel **getChannel**()

Returns the unique SocketChannel object associated with this socket, if any.

A socket will have a channel if, and only if, the channel itself was created via the SocketChannel.open or ServerSocketChannel.accept methods.

Returns:
the socket channel associated with this socket, or null if this socket was not created for a channel
Since: 1.4

---

getInputStream

public InputStream **getInputStream**()
throws IOException

Returns an input stream for this socket.

If this socket has an associated channel then the resulting input stream delegates all of its operations to the channel. If the channel is in non-blocking mode then the input stream's read operations will throw an IllegalBlockingModeException.

Under abnormal conditions the underlying connection may be broken by the remote host or the network software (for example a connection reset in the case of TCP connections). When a broken connection is detected by the network software the following applies to the returned input stream :-

- The network software may discard bytes that are buffered by the socket. Bytes that aren't discarded by the network software can be read using read.
- If there are no bytes buffered on the socket, or all buffered bytes have been consumed by read, then all subsequent calls to read will throw an IOException.
- If there are no bytes buffered on the socket, and the socket has not been closed using close, then available will return 0.

Returns:

an input stream for reading bytes from this socket.

Throws:
IOException - if an I/O error occurs when creating the input stream, the socket is closed, the socket is not connected, or the socket input has been shutdown using shutdownInput()

---

getOutputStream

public OutputStream **getOutputStream**()
throws IOException
Returns an output stream for this socket.

If this socket has an associated channel then the resulting output stream delegates all of its operations to the channel. If the channel is in non-blocking mode then the output stream's write operations will throw an IllegalBlockingModeException.

Returns:
an output stream for writing bytes to this socket.
Throws:
IOException - if an I/O error occurs when creating the output stream or if the socket is not connected.

---

setTcpNoDelay

public void **setTcpNoDelay**(boolean on)
throws SocketException
Enable/disable TCP_NODELAY (disable/enable Nagle's algorithm).
Parameters:
on - true to enable TCP_NODELAY, false to disable.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.
Since:
JDK1.1
See Also:
getTcpNoDelay()

---

getTcpNoDelay

public boolean **getTcpNoDelay**()
throws SocketException
Tests if TCP_NODELAY is enabled.
Returns:
a boolean indicating whether or not TCP_NODELAY is enabled.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since: JDK1.1
See Also:
setTcpNoDelay(boolean)

---

setSoLinger

public void **setSoLinger**(boolean on,int linger)throws SocketException

Enable/disable SO_LINGER with the specified linger time in seconds. The maximum timeout value is platform specific. The setting only affects socket close.
Parameters:
on - whether or not to linger on.
linger - how long to linger for, if on is true.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.
IllegalArgumentException - if the linger value is negative.
Since: JDK1.1
See Also:
getSoLinger()

---

getSoLinger

public int **getSoLinger**()throws SocketException

Returns setting for SO_LINGER. -1 returns implies that the option is disabled. The setting only affects socket close.
Returns:
the setting for SO_LINGER.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.
Since: JDK1.1
See Also:
setSoLinger(boolean, int)

---

sendUrgentData

public void **sendUrgentData**(int data)throws IOException

Send one byte of urgent data on the socket. The byte to be sent is the lowest eight bits of the data parameter. The urgent byte is sent after any preceding writes to the socket OutputStream and before any future writes to the OutputStream.
Parameters:
data - The byte of data to send
Throws:
IOException - if there is an error sending the data.
Since: 1.4

---

setOOBInline

public void **setOOBInline**(boolean on)throws SocketException

Enable/disable OOBINLINE (receipt of TCP urgent data) By default, this option is disabled and TCP urgent data received on a socket is silently discarded. If the user wishes to receive urgent data, then this option must be enabled. When enabled, urgent data is received inline with normal data. Note, only limited support is

provided for handling incoming urgent data. In particular, no notification of incoming urgent data is provided and there is no capability to distinguish between normal data and urgent data unless provided by a higher level protocol.
Parameters:
on - true to enable OOBINLINE, false to disable.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.
Since: 1.4
See Also:
getOOBInline()

---

getOOBInline

public boolean **getOOBInline**()throws SocketException

Tests if OOBINLINE is enabled.
Returns:
a boolean indicating whether or not OOBINLINE is enabled.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.
Since: 1.4
See Also:
setOOBInline(boolean)

---

setSoTimeout

public void **setSoTimeout**(int timeout)throws SocketException

Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds. With this option set to a non-zero timeout, a read() call on the InputStream associated with this Socket will block for only this amount of time. If the timeout expires, a **java.net.SocketTimeoutException** is raised, though the Socket is still valid. The option **must** be enabled prior to entering the blocking operation to have effect. The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout.
Parameters:
timeout - the specified timeout, in milliseconds.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.
Since: JDK 1.1
See Also:
getSoTimeout()

getSoTimeout

public int **getSoTimeout**() throws SocketException

Returns setting for SO_TIMEOUT. 0 returns implies that the option is disabled (i.e., timeout of infinity).

Returns:

the setting for SO_TIMEOUT

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

JDK1.1

See Also:

setSoTimeout(int)

---

setSendBufferSize

public void **setSendBufferSize**(int size)throws SocketException

Sets the SO_SNDBUF option to the specified value for this Socket. The SO_SNDBUF option is used by the platform's networking code as a hint for the size to set the underlying network I/O buffers.

Because SO_SNDBUF is a hint, applications that want to verify what size the buffers were set to should call getSendBufferSize().

Parameters:

size - the size to which to set the send buffer size. This value must be greater than 0.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

IllegalArgumentException - if the value is 0 or is negative.

Since: 1.2

See Also:

getSendBufferSize()

---

getSendBufferSize

public int **getSendBufferSize**()throws SocketException

Get value of the SO_SNDBUF option for this Socket, that is the buffer size used by the platform for output on this Socket.

Returns:

the value of the SO_SNDBUF option for this Socket.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since: 1.2

See Also:

setSendBufferSize(int)

setReceiveBufferSize

public void **setReceiveBufferSize**(int size)throws SocketException

Sets the SO_RCVBUF option to the specified value for this Socket. The SO_RCVBUF option is used by the platform's networking code as a hint for the size to set the underlying network I/O buffers.

Increasing the receive buffer size can increase the performance of network I/O for high-volume connection, while decreasing it can help reduce the backlog of incoming data.

Because SO_RCVBUF is a hint, applications that want to verify what size the buffers were set to should call getReceiveBufferSize().

The value of SO_RCVBUF is also used to set the TCP receive window that is advertized to the remote peer. Generally, the window size can be modified at any time when a socket is connected. However, if a receive window larger than 64K is required then this must be requested **before** the socket is connected to the remote peer. There are two cases to be aware of:

1. For sockets accepted from a ServerSocket, this must be done by calling ServerSocket.setReceiveBufferSize(int) before the ServerSocket is bound to a local address.
2. For client sockets, setReceiveBufferSize() must be called before connecting the socket to its remote peer.

Parameters:
size - the size to which to set the receive buffer size. This value must be greater than 0.
Throws:
IllegalArgumentException - if the value is 0 or is negative.
SocketException - if there is an error in the underlying protocol, such as a TCP error.
Since: 1.2
See Also:
getReceiveBufferSize(), ServerSocket.setReceiveBufferSize(int)

---

getReceiveBufferSize

public int **getReceiveBufferSize**()throws SocketException

Gets the value of the SO_RCVBUF option for this Socket, that is the buffer size used by the platform for input on this Socket.
Returns:
the value of the SO_RCVBUF option for this Socket.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.
Since: 1.2

See Also:
setReceiveBufferSize(int)

---

setKeepAlive

public void **setKeepAlive**(boolean on)throws SocketException

Enable/disable SO_KEEPALIVE.

Parameters:
on - whether or not to have socket keep alive turned on.

Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since: 1.3

See Also:
getKeepAlive()

---

getKeepAlive

public boolean **getKeepAlive**()throws SocketException

Tests if SO_KEEPALIVE is enabled.

Returns:
a boolean indicating whether or not SO_KEEPALIVE is enabled.

Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since: 1.3

See Also:
setKeepAlive(boolean)

---

setTrafficClass

public void **setTrafficClass**(int tc)throws SocketException

Sets traffic class or type-of-service octet in the IP header for packets sent from this Socket. As the underlying network implementation may ignore this value applications should consider it a hint.

The tc **must** be in the range 0 <= tc <= 255 or an IllegalArgumentException will be thrown.

Notes:

for Internet Protocol v4 the value consists of an octet with precedence and TOS fields as detailed in RFC 1349. The TOS field is bitset created by bitwise-or'ing values such the following :-

- IPTOS_LOWCOST (0x02)
- IPTOS_RELIABILITY (0x04)
- IPTOS_THROUGHPUT (0x08)

- IPTOS_LOWDELAY (0x10)

The last low order bit is always ignored as this corresponds to the MBZ (must be zero) bit.

Setting bits in the precedence field may result in a SocketException indicating that the operation is not permitted.

for Internet Protocol v6 tc is the value that would be placed into the sin6_flowinfo field of the IP header.

Parameters:
tc - an int value for the bitset.
Throws:
SocketException - if there is an error setting the traffic class or type-of-service
Since: 1.4
See Also:
getTrafficClass()

---

getTrafficClass

public int **getTrafficClass**()throws SocketException

Gets traffic class or type-of-service in the IP header for packets sent from this Socket

As the underlying network implementation may ignore the traffic class or type-of-service set using #setTrafficClass() this method may return a different value than was previously set using the #setTrafficClass() method on this Socket.

Returns:
the traffic class or type-of-service already set
Throws:
SocketException - if there is an error obtaining the traffic class or type-of-service value.
Since: 1.4
See Also:
setTrafficClass(int)

---

setReuseAddress

public void **setReuseAddress**(boolean on) throws SocketException

Enable/disable the SO_REUSEADDR socket option.

When a TCP connection is closed the connection may remain in a timeout state for a period of time after the connection is closed (typically known as the TIME_WAIT state or 2MSL wait state). For applications using a well known socket address or port it may not be possible to bind a socket to the required SocketAddress if there is a connection in the timeout state involving the socket address or port.

Enabling SO_REUSEADDR prior to binding the socket using bind(SocketAddress) allows the socket to be bound even though a previous connection is in a timeout state.

When a Socket is created the initial setting of SO_REUSEADDR is disabled.

The behaviour when SO_REUSEADDR is enabled or disabled after a socket is bound (See isBound()) is not defined.

Parameters:
on - whether to enable or disable the socket option
Throws:
SocketException - if an error occurs enabling or disabling the SO_RESUEADDR socket option, or the socket is closed.
Since: 1.4
See Also:
getReuseAddress(), bind(SocketAddress), isClosed(), isBound()

---

getReuseAddress

public boolean **getReuseAddress**()throws SocketException

Tests if SO_REUSEADDR is enabled.
Returns:
a boolean indicating whether or not SO_REUSEADDR is enabled.
Throws:
SocketException - if there is an error in the underlying protocol, such as a TCP error.
Since: 1.4
See Also:
setReuseAddress(boolean)

---

close

public void **close**()throws IOException

Closes this socket.

Any thread currently blocked in an I/O operation upon this socket will throw a SocketException.

Once a socket has been closed, it is not available for further networking use (i.e. can't be reconnected or rebound). A new socket needs to be created.

If this socket has an associated channel then the channel is closed as well.

Throws:
IOException - if an I/O error occurs when closing this socket.
See Also:
isClosed()

shutdownInput

public void **shutdownInput**()throws IOException

Places the input stream for this socket at "end of stream". Any data sent to the input stream side of the socket is acknowledged and then silently discarded.

If you read from a socket input stream after invoking shutdownInput() on the socket, the stream will return EOF.

Throws:
IOException - if an I/O error occurs when shutting down this socket.
Since: 1.3
See Also:
shutdownOutput(), close(), setSoLinger(boolean, int), isInputShutdown()

---

shutdownOutput

public void **shutdownOutput**()throws IOException

Disables the output stream for this socket. For a TCP socket, any previously written data will be sent followed by TCP's normal connection termination sequence. If you write to a socket output stream after invoking shutdownOutput() on the socket, the stream will throw an IOException.
Throws:
IOException - if an I/O error occurs when shutting down this socket.
Since:
1.3
See Also:
shutdownInput(), close(), setSoLinger(boolean, int), isOutputShutdown()

---

toString

public String **toString**()

Converts this socket to a String.
Overrides:
toString in class Object
Returns:
a string representation of this socket.

---

isConnected

public boolean **isConnected**()

Returns the connection state of the socket.
Returns:
true if the socket successfuly connected to a server
Since: 1.4

isBound

public boolean **isBound**()

Returns the binding state of the socket.

Returns:
true if the socket successfuly bound to an address
Since: 1.4
See Also:
bind(java.net.SocketAddress)

---

isClosed

public boolean **isClosed**()

Returns the closed state of the socket.
Returns:
true if the socket has been closed
Since: 1.4
See Also:
close()

---

isInputShutdown

public boolean **isInputShutdown**()

Returns wether the read-half of the socket connection is closed.
Returns:
true if the input of the socket has been shutdown
Since: 1.4
See Also:
shutdownInput()

---

isOutputShutdown

public boolean **isOutputShutdown**()

Returns wether the write-half of the socket connection is closed.
Returns:
true if the output of the socket has been shutdown
Since: 1.4
See Also:
shutdownOutput()

---

setSocketImplFactory

public static void **setSocketImplFactory**(SocketImplFactory fac)throws IOException

Sets the client socket implementation factory for the application. The factory can be specified only once.

When an application creates a new client socket, the socket implementation factory's createSocketImpl method is called to create the actual socket implementation.

If there is a security manager, this method first calls the security manager's checkSetFactory method to ensure the operation is allowed. This could result in a SecurityException.

Parameters:
fac - the desired factory.

Throws:
IOException - if an I/O error occurs when setting the socket factory. SocketException - if the factory is already defined.
SecurityException - if a security manager exists and its checkSetFactory method doesn't allow the operation.
See Also:
SocketImplFactory.createSocketImpl(), SecurityManager.checkSetFactory()

GLOSARIO DE TÉRMINOS.

Chat: Servicio de Internet basado en la comunicación en tiempo real y mediante teclado entre personas.

Chatear: Función que permite conversar en tiempo real y dentro de Internet entre personas situadas en distintos puntos del planeta mediante la utilización del teclado.

Chat Room: Espacio electrónico, un sitio Web o una sección de un servicio en línea, donde la gente puede comunicar en línea y en tiempo real.

Cliente: Un cliente es un programa que utiliza los servicios de otro programa. El programa cliente se utiliza para contactar y obtener datos u obtener un servicio a partir del servidor.

Dirección IP: Una dirección IP es un código numérico que identifica a un ordenador específico en Internet.

Emoticón: También denominados caritas, "smiley" en inglés,

Internet: El Internet es una "red de redes", y cualquiera puede intercambiar informaciones de manera fácil y libre.

Intranet: [De intra, interno y net, en inglés, red].- Red interna de una empresa, que parcialmente puede exponer información al exterior vía Internet.

IRC: Internet Relay Chat

Java: Lenguaje de programación multiplataforma.

LAN: Red de area Local

Protocolo: Es una serie de reglas que utilizan dos ordenadores para comunicar entre sí.

Puerto: Numero que identifica una entrada lógica en el ordenador.

Red: Una red, network en inglés, son dos o más ordenadores conectados entre sí de manera que puedan compartir recursos.

Socket: Son Mecanismos de comunicación entre programas a través de una red *TCP/IP.*

Servidor: Un servidor es un ordenador que trata las peticiones de datos, el Correo electrónico, la transferencia de ficheros, y otros servicios de red realizados por otros ordenadores (clientes).

TCP: Protocolo de control de transmisión.

TCP/IP: Son las siglas de Transmission Control Protocol/Internet Protocol, el lenguaje que rige todas las comunicaciones entre todos los ordenadores en Internet.

WAN: Red de area Ancha.

Word Wide Web: (en español "Telaraña Mundial") interfaz de comunicación en la Internet, que hace uso de enlaces de hipertexto en el interior de una misma página, o entre distintas páginas.