INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

"DESARROLLO DE UN MENSAJERO INSTANTÁNEO PARA EL INTERCAMBIO DE INFORMACIÓN ACADÉMICA EN LA ESIME ZACATENCO"

TESIS

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN COMUNICACIONES Y ELECTRÓNICA

PRESENTAN:

Carrillo Comparán Angel Adrian
Pérez Díaz Martha Angelica
Reyes Marcos Pedro Eduardo

Asesores:

M. en C. Zavala Mejía Genaro.
M. en C. Armando Martínez Ríos.

México, D.F. 2007

Índice de contenidos

Objetivo General

Desarrollar una aplicación de mensajería instantánea basado en el lenguaje de programación Java y en el protocolo de comunicaciones TCP/IP, el cual permitirá la comunicación entre usuarios y en el que se podrá acceder a salas de conversación de temas referentes a tópicos escolares para el apoyo a los cursos impartidos con ayuda de profesores u otros usuarios.

Objetivos Particulares

Desarrollar una herramienta sencilla y confiable que brinde apoyo extra para establecer comunicación entre miembros de la ESIME Zacatenco.

Realizar pruebas da la operación del mensajero instantáneo con una base de datos creada con los datos de los usuarios de la ESIME Zacatenco.

Realizar pruebas de campo en diferentes sistemas operativos, con la finalidad de brindar un buen servicio a todos los usuarios de esta unidad.

Justificación del Proyecto

Dentro de la Escuela Superior de Ingeniería en Comunicaciones y Electrónica unidad Zacateco, nos encontramos con la necesidad de hacer preguntas de manera externa debido a que muchas veces los temas expuestos en clase por los profesores y/o alumnos no quedan del todo claros, es necesario que se repasen estos temas, pero muchas veces debido a falta de tiempo o a la carga de trabajo que se deja en las demás asignaturas, entre otras, no es posible ir directamente con el profesor o con algún asesor para resolver las dudas que hayan surgido, por este motivo se ha propuesto diseñar este software orientado a asesorías en línea para que cualquier alumno pueda ingresar y preguntar sobre los temas previamente vistos y así algún profesor u otro alumno puedan ayudar a solventar estas dudas.

De la misma manera, en diversas ocasiones el sistema administrativo que maneja el instituto, no es lo suficientemente eficiente para resolver todas nuestras dudas acerca de nuestro estado académico y sobretodo de recibir una buena atención en los tiempos de atención a alumnos, así que se considera que sería una buena opción la utilización de este software.

Esta aplicación difiere de otras aplicaciones ya que ocupa poco espacio en memoria, es un software libre de virus, es de uso exclusivo para la ESIME Zacatenco, es de fácil instalación y uso.

Introducción

Innovación en el manejo de los recursos para el aprendizaje

El avance de la tecnología educativa y la nueva visión acerca de los procesos de enseñanza-aprendizaje, llevan a la educación a una modalidad de ambientes de aprendizaje apoyado por los medios, diseñados para crear condiciones pedagógicas y contextuales favorables, donde el conocimiento y sus relaciones con los individuos son el factor principal para formar una "sociedad del conocimiento".

En un entorno virtual con apoyo de medios tecnológicos, en la representación aproximada de la realidad se puede construir un entorno más amplio en el que además del aula y la escuela la simulación incluya el contexto social que redunda en una significatividad más amplia para el estudiante, y que también constituye un ambiente de aprendizaje.

En la actualidad, se dispone de diversos recursos tecnológicos denominados de punta que facilitan las tareas e impulsan la productividad en la educación. El empleo de medios de comunicación como el cable telefónico, el coaxial, la fibra óptica y las transmisiones vía satélite promueven la disponibilidad de información gracias al incremento notable en velocidad, capacidad y versatilidad de plataformas electrónicas.

Con el progreso tecnológico se entrelazan las virtudes de los equipos, los nuevos y más amigables ambientes virtuales y la capacidad de transmisión. Se cuenta con medios que invitan al desarrollo creativo de aplicaciones en pro de la educación, la multimedia, los programas interactivos de simulación, el correo electrónico, los documentos virtuales, las líneas de discusión en tiempo real (chats) y asincrónicos (foros); opciones que, orientadas principalmente a la educación a distancia, en el camino se acoplan de manera interesante e innovadora a su empleo en la enseñanza presencial.

En sí, esta propuesta incluye la incorporación de un mensajero instantáneo que permita la interacción en red entre alumnos y maestro. ***El mensajero instantáneo*** propone desencadenar interacciones entre los estudiantes, ellos pueden estar en constante comunicación entre sí o con el maestro para formular nuevas ideas o planteamientos de problemas, así como también dar solución a éstos. Es una herramienta que puede ser utilizada como parte de las otras herramientas o con un uso individual.

Para diseñar el software , en un principio, es necesario escoger el lenguaje bajo el que se va a desarrollar, para esta eleccion se han tomado en cuenta diferentes lenguajes, entre estos destacan por su poder de aplicación C++, C#, y Java.

Se ha escogido Java por varias razones, algunas de las cuales se enlistan a continuacion:

- Aunque C++ es un lenguaje orientado a objetos, este presenta caracteristicas de un lenguaje estructurado, ya que esta basado en C, por el contrario C# y Java son lenguajes Orientados a Objetos en su totalidad, por lo que la programación es mas sencilla y pura.

- Java al contrario de C++ es un lenguaje multiplataforma, esto quiere decir que es independiente del Sistema Operativo en el que se ejecute, y aunque en la actualidad se tienen grandes avances en el proyecto Mono, que permite la integración de la tecnología. NET en plataformas abiertas, los Bytecodes generados por un programa escrito en Java compilado en un Sistema Operativo Windows pueden ser transferidos a un Sistema

Operativo diferente, como Linux, y ejecutados sin la necesidad de realizar alguna modificación.

- Java permite implementar una API para la programación en red, por lo que es más sencillo comunicarse mediante el protocolo HTTP, y con sockets. C# implementa una API que de igualmanera incorpora de manera sencilla, al contrario de C++ el cual requiere de un mayor esfuerzo para poder realizar una comunicación con sockets.

- Dado a su naturaleza, Java cuenta con un gran numero de seguidores y comunidades especializadas en este lenguaje, por lo que es sencillo consultar alguna de estas para aclarar dudas sobre la programacion.

- Java dispone de una gran cantidad de IDE's de codigo abierto que pueden ayudar a facilitar la programacion, algunos ejemplos de estos son NetBeans y Eclipse.

Los puntos anteriores representan algunas de las ventajas que se tienen al programar bajo el lenguaje de Java, y por estas razones nos hemos decidido a realizarlo bajo esta plataforma.

Dado que solo se transmitiran mensajes nos centraremos en el funcionamiento de los protocolos IP, TCP y UDP, ya que estos son protocolos altamente usados y pueden cubrir completamente nuestras necesidades.

El protocolo IP es el protocolo usado para enviar datos entre las PC´s conectadas a Internet o a una intranet. Su funcion consiste en enviar datos a traves de paquetes.

Dado que IP es un protocolo sin conexión no envia informacion de control, esto provoca que no haya una garantia de que los paquetes se entreguen en el orden en el que fueron enviados y estos pueden llegar con errores.

Para realizar la comunicación entre aplicaciones TCP/IP contiene dos protocolos de transporte (TCP y UDP).

El protocolo TCP es un servicio de transporte orientado a la conexión. Este protocolo incorpora algoritmos de deteccion de errores, con lo que se garantiza que los paquetes seran entregados en el mismo orden con el que salieron. Esta deteccion de errores tiene un problema, ya que hay que enviar informacion redundante para comprobar que se mande correctamente la informacion se requiere de un mayor tiempo para enviar la informacion.

El protocolo UDP es un servidor de transporte sin conexión. Este protocolo descompone los mensajes en paquetes y son enviados de manera independiente unos de otros. Lo que provoca que los paquetes puedan llegar en un orden diferente al que se enviaron y el receptor no puede saber si llegaron correctamente o no. La ventaja de este protocolo es que es mas rapido que TCP.

Ambos protocolos usan sockets para comunicar terminales en una forma de cliente-servidor. Los sockets se dividen en dos:

- Socket activo. Puede enviar y recibir datos a través de una conexión abierta.

- Socket pasivo. Esperan intentos de conexión. Cuando llega una conexión entrante se le asigna un socket activo. No sirven para enviar o recibir datos.

Como ya se ha mencionado Java cuenta con un paquete orientado a la conexión. A este paquete se le llama java.net.

Este paquete permite crear sockets, conexiones HTTP, etc; y esto va a ser necesario para poder realizar nuestro proyecto. Este paquete comprende clases que se pueden dividir en:

- Clases que corresponden a las API de los sockets: Socket, ServerSocket, DatagramSocket.

- Clases correspondientes a herramientas para trabajar con URL: URL, URLConnection, HttpURLConnection, URLEncoder.

El paquete java.net permite trabajar con los protocolos TCP y UDP, y cada uno cuenta con sus propios metodos para crear sockets y poder realizar la comunicación entre cliente-servidor.

Para el desarrollo de esta aplicación sera necesario crear una aplicacion servidor, la cual organizará el acceso y realizará un historial de las conexiones realizadas por los usuarios a la base de datos; también sera necesario realizar una aplicación cliente, con la cual los usuarios podrán realizar la conección con la aplicación servidor y asi comunicarse con los demas usuarios y entrar a las salas referentes a las materias.

Como ya se mencionó anteriormente se necesitará de una base de datos para guardar la información de cada usuario y asi poder llevar un mejor control sobre los usuarios. Se decidió usar MySQL por que es una base de datos que trabaja con la licencia GNU GPL, la cual es una licencia de software libre que permite el uso personal sin restricciones, ademas de poseer Drivers para establecer la conexión desde varios lenguajes de programación, entre los que se incluye Java.

En resumen, en esta parte de introducción se habló básicamente de los siguientes puntos:

- La descripción general del proyecto.
- Los recursos de aprendizaje virtual.
- El lenguaje de programación y protocolo a usar.

En este trabajo básicamente está compuesto por seis capítulos. El primer capítulo llamado principios teóricos describe los fundamentos que implica todo el desarrollo del proyecto. El segundo consiste en la justificación de las necesidades que nos indican que tan factible y viable es el proyecto. El tercer capítulo habla del proceso que se llevo a cabo para el desarrollo de la aplicación y las herramientas que intervinieron en su diseño. El cuarto capítulo consiste en el análisis de los costos involucrados en el diseño y el desarrollo del proyecto basado en costos de producción reales. El capítulo cinco se presentan los resultados de cada una de las pruebas realizadas en diferentes entornos de trabajo. El sexto y último capítulo consiste en las conclusiones y recomendaciones en las que se finaliza todo el trabajo realizado y los objetivos alcanzados en base a los resultados obtenidos.

Capítulo I

Principios Teóricos

En este capítulo se hace una breve descripción de los principios teóricos que fundamentarán este proyecto:

- Descripción de la Mensajería Instantánea.
- Descripción de Java.
- Cliente y Servidor.
- Sockets.
- Bases de Datos.

1.1 Mensajeros Instantáneos

La mensajería instantánea (conocida también en inglés como IM) requiere el uso de un cliente informático que realiza el servicio de mensajería instantánea y que se diferencia del correo electrónico en que las conversaciones se realizan en tiempo real. La mayoría de los servicios ofrecen el "aviso de presencia", indicando cuando el cliente de una persona en la lista de contactos se conecta o en qué estado se encuentra, si está disponible para tener una conversación. En los primeros programas de mensajería instantánea, cada letra era enviada según se escribía y así, las correcciones de las letras también se veían en tiempo real. Esto daba a las conversaciones más la sensación de una conversación telefónica que un intercambio de texto. En los programas actuales, habitualmente, se envía cada frase de texto al terminarse de escribir. Además, en algunos, también se permite dejar mensajes aunque la otra parte no esté conectada al estilo de un contestador automático. Otra función que tienen muchos servicios es el envío de archivos.

La mayoría usan redes propietarias de los diferentes softwares que ofrecen este servicio. Adicionalmente, hay programas de mensajería instantánea que utilizan el protocolo abierto Jabber, con un conjunto descentralizado de servidores.

Los mensajeros instantáneos más utilizados son ICQ, Yahoo! Messenger, MSN Messenger, AIM (AOL Instant Messenger) y Google Talk (que usa el protocolo abierto Jabber). Estos servicios han heredado algunas ideas del viejo, aunque aún popular, sistema de conversación IRC. Cada uno de estos mensajeros permite enviar y recibir mensajes de otros usuarios usando los mismos software clientes, sin embargo, últimamente han aparecido algunos clientes de mensajerías que ofrecen la posibilidad de conectarse a varias redes al mismo tiempo (aunque necesitan registrar usuario distinto en cada una de ellas). También existen programas que ofrecen la posibilidad de conectarse a varias cuentas de usuario a la vez como MSN.

Los sistemas de mensajería tienen unas funciones básicas aparte de mostrar los usuarios que hay conectados y poder establecer una conversación. Unas son comunes a todos o casi todos los clientes o protocolos y otras son menos comunes:

Contactos:

- Mostrar varios estados: disponible, disponible para hablar, sin actividad, no disponible, vuelvo enseguida, invisible, no conectado.

- Mostrar un mensaje de estado: Es una palabra o frase que aparece en las listas de contactos de tus amigos junto a tu nick. Puede indicar la causa de la ausencia, o en el caso del estado disponible para hablar, el tema del que quieres hablar.

- Registrar y borrar usuarios de la lista de contactos propia.

- A veces se pueden agrupar los contactos: familia, trabajo, facultad, etc.

- Se puede usar un avatar: una imagen que le identifique a uno. No tiene por qué ser la foto de uno mismo.

- Conversación:
  Puede haber varios tipos de mensajes:
  - Aviso: Lanza un mensaje solo. No es una invitación a mantener la conversación, solo se quiere enviar una información. Un ejemplo de uso de este tipo sería el Mensaje del día (mensaje personal).
  - Invitación a conversar: Se invita a mantener una conversación tiempo real.
  - Mensaje emergente: Es un aviso que se despliega unos segundos y se vuelve a cerrar. No requiere atención si no se desea. Sirve como aviso breve que moleste lo mínimo posible. Por ejemplo, "ya lo encontré, gracias"

- Muchas veces es útil mostrar cuando el otro está escribiendo.

- Muchas veces se puede usar emoticones (imágenes usadas para expresar estados de ánimo).

Conversaciones en grupo:

- Se pueden crear salas (grupos de conversación), públicas y privadas y también permanentes o que desaparezcan al quedarse sin usuarios.

- Restringir el acceso a salas mediante invitaciones certificadas, para invitar solo a quien uno quiera.

Otras:

- Mandar archivos.

- Posibilidad de usar otros sistemas de comunicación, como una pizarra electrónica, o abrir otros programas como una videoconferencia.

1.2 Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los 90's. A diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, Java es compilado en un bytecode que es interpretado (usando normalmente un compilador JIT), por una máquina virtual Java.

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos mucho más simple y elimina herramientas de bajo nivel como punteros.

Java está sólo lejanamente emparentado con JavaScript, aunque tengan nombres similares y compartan una sintaxis al estilo de C algo parecida.

Es otra manera de incluir código a ejecutar en los clientes que visualizan una página web. Se trata de pequeños programas hechos en Java, que se transfieren con las páginas web y que el navegador ejecuta en el espacio de la página.

Java es compatible con todos los sistemas porque basa su funcionamiento en los bytecodes, que no es más que una pre-compilación del código fuente de Java.

Estos bytecodes no son el programa en Java propiamente dicho, sino un archivo que contiene un código intermedio que puede manejar la Máquina Virtual de Java. Cada sistema operativo dispone de una Maquina Virtual de Java que puede interpretar los bytecodes y transformarlos a sentencias ejecutables en el sistema en cuestión.

1.2.1 J2SE

Con la aparición de J2SE (Java 2 Standard Edition – Java 2 edición estándar) los desarrolladores pudieron contar con las API (Application Programming Interface – interfaz de programación de aplicaciones) necesarias para construir aplicaciones Java robustas, pero esta edición aún no resolvía las necesidades requeridas por el muido empresarial en cuanto a las aplicaciones distribuidas se refiere (un ambiente computacional se dice distribuido cuando sus aplicaciones o bases de datos están ubicados en dos o más computadores). Así que Sun Microsystems, empujada por los sistemas corporativos, sintió la necesidad de revisar la tecnología Java. El resultado fue J2EE (Java 2 Enterprise Edition – Java 2 edición empresarial), versión ampliada de la J2SP que incluye las API necesarias para construir aplicaciones para arquitecturas distribuidas, permitiendo la construcción de sistemas transaccionales basados en tecnologías Web.

Las aplicaciones J2EE están hechas a base de componentes. Un componente J2EE es una unidad de software independiente que es ensamblado en la aplicación junto con sus archivos y clases, y que se comunica con otros componentes a través de canales definidos. La especificación J2EE define los siguientes componentes:

- *Aplicaciones cliente.* Son componentes que se ejecutan en el cliente. Una aplicación cliente normalmente se utiliza para mostrar una interfaz gráfica al usuario

- *Servlets y Java Server Pages (JSP).* Son componentes Web que se ejecutan en un servidor J2EE, concretamente en el contenedor del mismo. Los servlets son clases escritas en Java que procesan peticiones y construyen respuestas dinámicamente; por lo general un servlet obtiene datos como resultado de procesar la petición, que envía una página JSP para que se los transmita al cliente. Y las páginas JSP son documentos de texto que se ejecutan como los servlets, pero proporcionan una forma más natural de crear contenido dinámico.

- *Enterprise JavaBeans (EJB) o simplemente Enterprise beans.* Son componentes que se ejecutan en un servidor J2EE, concretamente en el contenedor EJB del mismo, y que contiene las reglas del negocio. *Hay tres clases de Enterprise beans: beans de sesión (sesión beans), beans de entidad (entity beans) y beans orientados a mensajes (message-driven beans).* Un bean de sesión representa una conversación transitoria con un cliente;

cuando termina la ejecución del cliente, el bean de sesión finaliza sin guardar sus datos. Por el contrario, un bean de entidad representa datos almacenados en una base de datos; si el cliente termina o se cierra el servidor, los servicios subyacentes aseguran que los datos del bean se guarden. Un bean orientado a mensajes combina características de un bean orientado a mensajes combina características de un bean de sesión y un oyente de mensajes JMS (Java Message Service), permitiendo que un componente de negocio recibir mensajes JMS de forma asíncrona, esto se muestra en la figura 1.1.

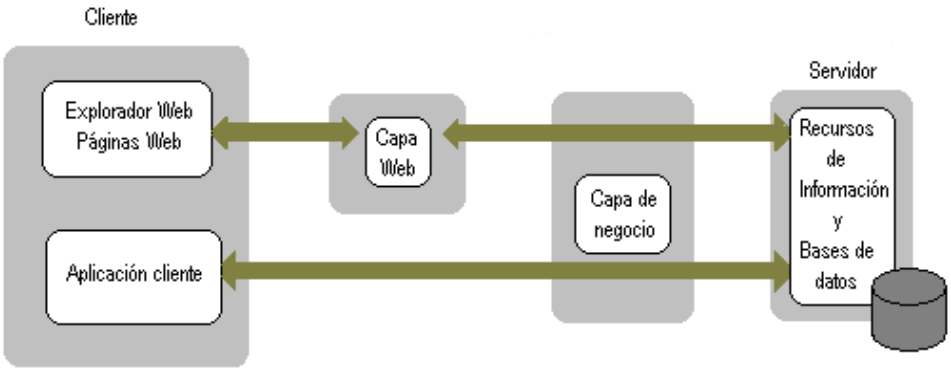


Figura 1.1 Componentes de J2SE

1.2.2 Contenedores J2SE

Un contenedor es una interfaz entre un componente y el sistema de nivel inferior que da soporte al componente. Mientras que un contenedor se encarga de la persistencia de los datos, la gestión de recursos, la seguridad, los hilos y otros servicios a nivel se sistema para los componentes asociados con él, estos son responsables de implementar la lógica del negocio. Esto significa que durante el desarrollo de una aplicación el programador puede concentrar todos sus esfuerzos en codificar las reglas de negocio, si tener que preocuparse de los servicios del sistema.

Resumiendo, un cliente se refiere a un programa que solicita un servicio de un componente. Un componente es, a su vez, otro programa que realiza una función para proporcionar el servicio; para ello, en algunos casos, puede necesitar obtener algún recurso. Y un contenedor es el software que gestiona al componente y le proporciona servicios de sistema.

1.2.3 Cliente

El software del cliente, normalmente tiene como misión:

- *Servir de interfaz gráfica al usuario para interactuar con la aplicación*. Esta interfaz normalmente estará basada en un explorador o en una aplicación cliente. El uso de un explorador, también llamado navegador, facilita la escritura de la interfaz por que los controles que componen la misma y la gestión de eventos vienen como parte del explorador, pero hay que asegurarse que la presentación es aceptable en distintos exploradores y versiones de los mismos, ya que es el explorador el que controla la forma en la que se muestran los controles. Por otra parte, el diseño de la interfaz está limitado a las características que se puedan describir con el lenguaje HTML (Hypertext Markup Languaje) o XML (Extensible Markup Languaje – lenguaje extensible para análisis de

documentos) utilizado, y su ejecución depende del servidor; esto es, el explorador debe llamar al servidor prácticamente cada vez que ocurre un evento en la interfaz, lo que repercute desfavorablemente en el rendimiento y aumenta el tiempo de respuesta. En cambio una aplicación cliente, aunque sea más difícil de escribir, permite un control total sobre los elementos de la interfaz y la gestión de eventos; esto permitirá programar que el servidor sea llamado sólo cuando sea necesario lo que repercutirá favorablemente en el rendimiento final de la aplicación.

- *Requerir información del usuario y validarla.* Los detalles del proceso de validación dependen de la aplicación y éste puede ser realizado en el cliente, y siempre en el servidor. El proceso de validación puede minimizarse en la medida que utilicemos controles tales como botones de opción, casillas de verificación, listas, etc., que proporcionan entradas fijas y, por lo tanto, no necesita validación.

Permite construir componentes, que contengan lógica de negocio o que representen datos, reutilizables por distintas aplicaciones cliente y que normalmente estén distribuidos entre distintas máquinas.

1.2.4 Servidor

Tradicionalmente, los sistemas corporativos se han venido diseñando utilizando el modelo cliente servidor, entendiendo por cliente una aplicación que inicia en el diálogo con otra denominada servidor para solicitarle servicios que éste puede atender. En la figura 1.2, se ilustra cómo puede imaginarse este modelo:

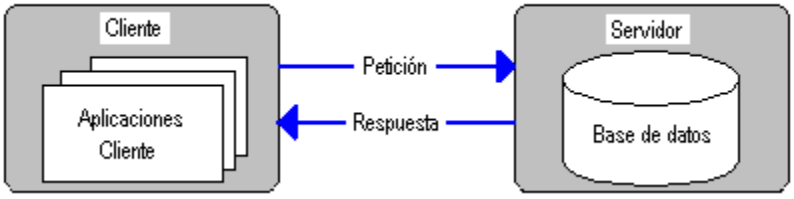


Figura 1.2 Funcionamiento cliente-servidor

Una aplicación cliente servidor tiene tres componentes fundamentales: presentación (interfaz de usuario) lógica de negocio y gestión de datos. Cada una de estas componentes puede estar en el cliente o en el servidor.

Actualmente, la lógica de negocio se ha convertido también en un servicio y puede recibir en cualquier otro servidor, conocido como servidor de aplicaciones, dando lugar a una arquitectura de tres capas (figura 1.3).

Figura 1.3 Aplicación servidor

Una aplicación diseñada según el modelo de tres capas, se divide en presentación, lógica de negocio y datos. La capa de presentación contiene todos los elementos que constituyen una interfaz con el usuario. Esta capa incluye todo aquello con lo que el usuario puede interactuar, por ejemplo, una interfaz gráfica basada en ventanas, un explorador, también llamado navegador, etc. En la capa lógica de negocio, se modela el comportamiento del sistema, basándose en los datos provistos por la capa de datos, y actualizándolos según sea necesario. Esta capa describe los distintos cálculos y otros procesos a realizar con los datos.

Finalmente, la capa de datos representa el mecanismo para el acceso y el almacenamiento de la información. Consiste generalmente en un gestor de bases de datos o de objetos, y el esquema de datos propios de cada aplicación.

1.3 Sockets

Los sockets son puntos finales de enlaces de comunicaciones entre procesos. 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.

El tipo de sockets describe la forma en la que se transfiere información a través de ese socket.

1.3.1 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.

1.3.2 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.

1.3.3 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.

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 orientado a no conexión, 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 orientado a conexión, 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.

1.4 Bases de Datos

Una base de datos es un sistema para archivar información en computadora cuyo propósito general es mantener información y hacer que esté disponible cuando se solicite.

Las bases de datos son un área de la computación que ha recibido mucha atención debido a sus aplicaciones múltiples: bibliotecas, automatización de oficinas, ingeniería de software, diccionarios automatizados y en general cualquier programa orientado a mantener y recuperar información textual. Su recuperación, actualización y manejo es relativamente simple con el uso de cualquier manejador de bases de datos.

Cuando hablamos de documentos con estructura nos estamos refiriendo a documentos cuya estructura es declarada explícitamente de algún modo, asociando etiquetas a elementos de la estructura o mediante la sintaxis con la que se escribe el documento, como se hace en los lenguajes de programación.

Actualmente, el estudio de las bases de datos no estructuradas se ha acentuado por diversas causas. Una de las principales, es la recuperación de información. El objetivo de cualquier sistema relacionado con recuperación de información es ayudar a los usuarios de la base de datos a encontrar lo que buscan. Una base de datos textual es aquella cuyo propósito es almacenar documentos de texto, que pueden ser estructurados o no.

Conocemos varios métodos de recuperación de información, uno de ellos es la búsqueda directa o búsqueda tradicional. Los algoritmos tradicionales para este tipo de búsqueda pueden ser secuenciales y en orden, de acceso aleatorio (hashing), binarios (árboles balanceados o Skip Lists). La búsqueda directa se puede parafrasear así: requiere de un dato de entrada al que llamamos llave (o clave) que nos sirve para accesar a un documento. El programa nos enviará el documento al que corresponde la llave. En caso de no encontrarla enviará una notificación o simplemente nada. Otra manera de recuperar información es la búsqueda por contenido.

1.4.1 Las Bases de Datos Relacionales

En los esquemas tradicionales, las bases de datos relacionales son las que han tenido más uso comercial, consisten de tablas, cada tabla lleva el nombre de una relación y contiene filas y columnas. Nuevas tablas pueden ser construidas a partir de estas mediante el corte y pegado de filas y columnas (registros y campos) a partir de tablas existentes. Estas filas y columnas son lo que conocemos como campos y registros. La manera de accesar a cualquiera de ellos se realiza mediante un campo llave; éste se elige después de analizar todos los campos del registro.

La característica esencial de un campo llave es que es distinto para cada registro. Al procedimiento de seleccionar el campo llave y en general la estructuración de la base de datos se le llama normalización.

El campo llave sirve para localizar un registro dado y eventualmente seleccionar alguno de sus campos. El hecho de que sea no repetido permite utilizar búsqueda binaria para localizar un registro. Si quisiéramos buscar un campo del registro que no es el campo llave la búsqueda tendría que realizarse de manera secuencial, o bien construir una tabla alternativa en la que el campo llave sea el que nos interesa.

1.4.2 Bases de Datos Orientadas a Objetos

Las bases de datos orientadas a objetos, tienen una organización similar a la de los árboles. Donde cada nodo del árbol representa un campo y cada árbol un registro, cada tipo de nodo tiene un método distinto de búsqueda. Que es equivalente a decir que todos los campos pueden ser utilizados como campos llave, pero complica el diseño. Si la base de datos es demasiado grande, o tiene relaciones demasiado complejas el grafo resultante se vuelve una maraña ininteligible. No existen implementaciones comerciales de este tipo de bases de datos, y los prototipos académicos son una capa de software adicional al manejador de una base de datos relacional.

1.4.3 Bases de Datos Lógicas

Las bases de datos lógicas son construidas con registros homogéneos de manera parecida a las relacionales. Adicionalmente se agregan restricciones lógicas y reglas de composición parecidas a las de Prolog, que permiten deducir información que originalmente no está contenida en la base de datos.

1.4.4 Repositorios de información

Consisten en grandes volúmenes de datos que contienen documentos de texto o imágenes, representan un caso de bases de datos no estructuradas. El usuario localiza documentos y objetos haciendo una descripción de ellos.

1.5 JAVA.NIO

Como sabemos, Java es un lenguaje orientado a red, cuyo potencial radica en todo aquello que tengo que ver con la ejecución de aplicaciones a través de la red y. por tanto, este paquete se ajusta perfectamente a esta condición permitiendo la gestión de archivos desde sitios remotos.

Java.nio (New Input/Output), es una nueva librería de entrada/salida introducida con la versión 1.4 de Java. Ofrece una nueva API (interfaz de programación de aplicaciones) de entrada y salida (E/S).

La API de NIO fueron diseñados para proporcionar acceso a los niveles bajos de operaciones de E / S de los sistemas operativos modernos. Aunque las API son relativamente de alto nivel, el propósito es facilitar una aplicación que puede utilizar directamente la más eficiente las operaciones de la plataforma subyacente.

El paquete java.nio define las clases de amortiguación, que se utilizan en toda la API de NIO. La API de caracteres se define en el `java.nio.charset` paquete, y el selector de canales y APIs están definidas en el paquete `java.nio.channels`. Cada uno de estos subpaquetes tiene su propio proveedor de servicio (SPI), cuyo contenido puede ser utilizado para ampliar la plataforma por defecto de la implementación o de construir implementaciones alternativas.

Esta nueva aplicación puede dividirse en tres grandes paquetes:

Java.nio define **`buffers`**, que se usan para almacenar secuencias de bytes o de otros valores primitivos (int, float, char, doublé, etc.)

`Java.nio.channels` define canales, esto es, abstracciones mediante los cuales pueden transferirse datos entre los buffers y las fuentes o los consumidores de datos (un socket, un archivo, un dispositivo de hardware). Además, proporciona clases que permiten E/S sin bloqueo.

`Java.nio.charset` contiene clases que convierten de manera muy eficaz buffers de bytes en bytes de caracteres y que permiten el uso de expresiones regulares.

La NIO API incluye las siguientes características;

- Bufferes de datos de tipo primitivo.
- Conjunto de caracteres de los codificadores y decodificadores.
- Un archivo de interfaz que soporta cerraduras y cartografia de la memoria.
- Un multiplex, no bloqueo de E/S para la instalación de servidores escalables escrito.
- Incorpora buffers (contenedores de datos), canales (encargados de la transferencia de datos entre los buffers y las peticiones de E/S) y selectores (Encargados de proporcionar el estado de los canales: si están preparados para leer, escribir, recibir conexiones).
- Permite la transferencia eficaz de grandes cantidades de datos empleando bloques de datos.
- Los sockets de NIO permiten trabajar sin bloqueo (también permiten bloqueo). Sin bloqueo, un solo hilo puede encargarse de controlar tantos canales como se precise. No se necesita destinar un hilo a cada canal, lo cual aumenta el rendimiento de la E/S. Así mismo, al no necesitarse un hilo por socket, la complejidad del código se reduce notablemente. Más trabajo para el sistema operativo y la máquina virtual de Java, menos para el programador.

- Aprovecha las prestaciones del sistema operativo donde se ejecuta la MVJ.

1.5.1 Clase java.nio.Buffer

La clase `java.nio.Buffer` es una clase abstracta que tiene siete implementaciones (ver figura 1.4):

- `Java.nio.ByteBuffer`
- `Java.nio.CharBuffer`
- `Java.nio.IntBuffer`
- `Java.nio.LongBuffer`
- `Java.nio.ShortBuffer`
- `Java.nio.FloatBuffer`
- `Java.nio.Double Buffer`

Cada una de estas clases es un contenedor de datos primitivos con una capacidad limitada. En cada una, los datos se almacenan de manera lineal y secuencial. En comparación con las colecciones de Java, los buffers evitan la sobrecarga de la ubicación de los objetos en la pila y la recolección de basura.

Las dos maneras más frecuentes de crear un buffer son mediante los métodos estáticos `allocate(int capacidad)` y `allocateDirect(int capacidad)`. El segundo permite crear buffers directos, esto es, buffers en los que la máquina virtual de java intentará no usar memorias de almacenamiento temporal entre la MVJ y el sistema operativo en cada llamada a alguna operación específica del SO.

En un buffer, la capacidad es el número de elementos que puede contener, el límite, es el índice del primer elemento que no puede leerse o escribirse; la posición es el índice del próximo elemento que va a leerse o escribirse. La posición y el límite son menores o iguales a la capacidad, y la posición no puede ser mayor que el límite.

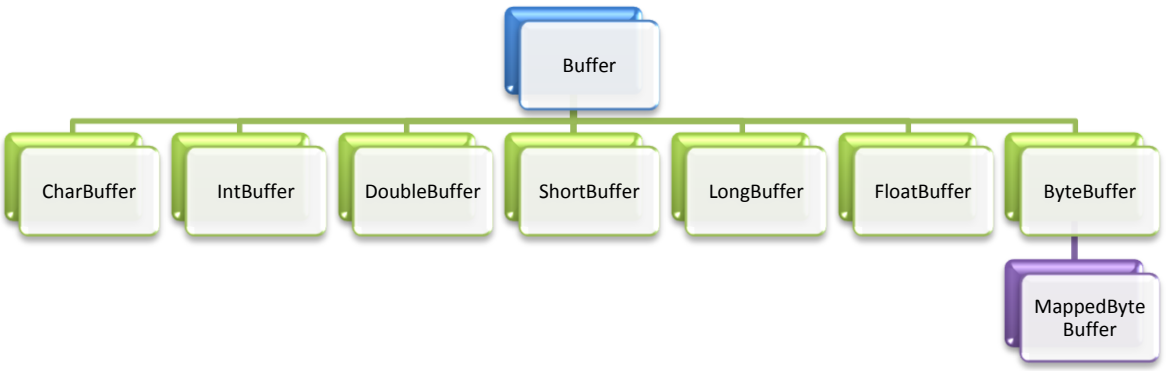


Figura 1.4 Jerarquía de clases del paquete Java.nio.buffer

Hay tres métodos que deben conocerse para realizar operaciones de lectura y escritura con buffers:

1) `public final Buffer clear()` limpia el buffer. La posición se iguala a cero y el límite de la capacidad.

2) `public final Buffer flip()` da la vuelta al buffer. El límite se establece en la posición actual y la posición se iguala a caro. Tras una llamada a este método, el buffer queda preparado para que los canales puedan leer sus datos.
3) `public final Buffer rewind()` rebobina el buffer: la posición se iguala a cero. Tras una llamada a este método, el buffer queda listo para que sus datos puedan escribirse en cualquier canal.

Cada clase Buffer cuenta con un método `public final int Buffer remaining()`, que devuelve el número de elementos que quedan por leer o por escribir, es decir, entre las posición actual y el límite; `public final boolean hasRemaining()` nos permite saber si existen elementos en la posición actual y el límite.

Así mismo, cada una de estas clases tienen sus métodos `put()` y `get()`, que permiten leer o escribir un dato del tipo correspondiente (char, int, etc.).

1.5.2 Los canales

Un canal, representa una conexión abierta a una entidad como un dispositivo de hardware, un archivo, un socket de red o un componente de software que es capaz de realizar una o más operaciones distintas de E/S. Un canal está abierto tras su creación y una vez cerrado permanece cerrado. Una vez que un canal está cerrado, cualquier intento de llamar a una operación de E/S sobre él, causará que se arroje una `ClosedChannelException`.

Dicho de otra manera, un canal es una conexión entre un buffer y una fuente o un consumidor de datos (un socket, un archivo, etc.). los datos pueden leerse de los canales mediante buffers y los datos de un buffer pueden escribirse en los canales. Es decir, los buffers pueden escribirse en canales o escribirse en estos.

Los canales pueden operar con bloqueo o sin él. Una operación de E/S con bloqueo no retorna hasta que se produce una de estas situaciones: a) se completa la operación; b) se produce una interrupción debida al sistema operativo; c) se lanza una excepción. Todos los métodos `read()` y `write()` no solo pueden producir bloqueos en esos casos.

Una operación de E/S sin bloqueo retoma al instante, devolviendo algún valor de retorno que indica si la operación se realizó correctamente o no. Un programa o un hilo que ejecute una operación sin bloqueo no se quedará dormido esperando datos, una interrupción o una excepción; su curso de ejecución continuará.

Los buffers son intermediarios entre los canales. No es posible pasar directamente datos de un canal a otro. Para leer datos de un canal y escribirlos en otro, hay que leer primero a un buffer y luego escribir del buffer al segundo canal. (ver figura 1.5).

Figura 1.5. Funcionamiento de los canales.

1.5.3 Clase java.nio.channels.Selector

La clase `java.nio.channels.Selector` es una de las principales de la API NIO. Un objeto selector controla una serie de canales y lanza un aviso cuando uno de ellos lanza un suceso de E/S. La clase Selector informa a la aplicación de las operaciones de E/S que ocurren en los canales que ésta mantiene activos.

La información sobre las operaciones e E/S se registra en un conjunto de claves, que son instancias de la clase `java.nio.channels.SelectionKey`. Cada clave almacena información sobre el canal que desencadena la operación y el tipo de ella (lectura, escritura, conexión entrante, conexión aceptada) (ver figura 1.6).

En la clase Selector, las instancias se crean con el método estático `public static Selector Open() throws IOException`.

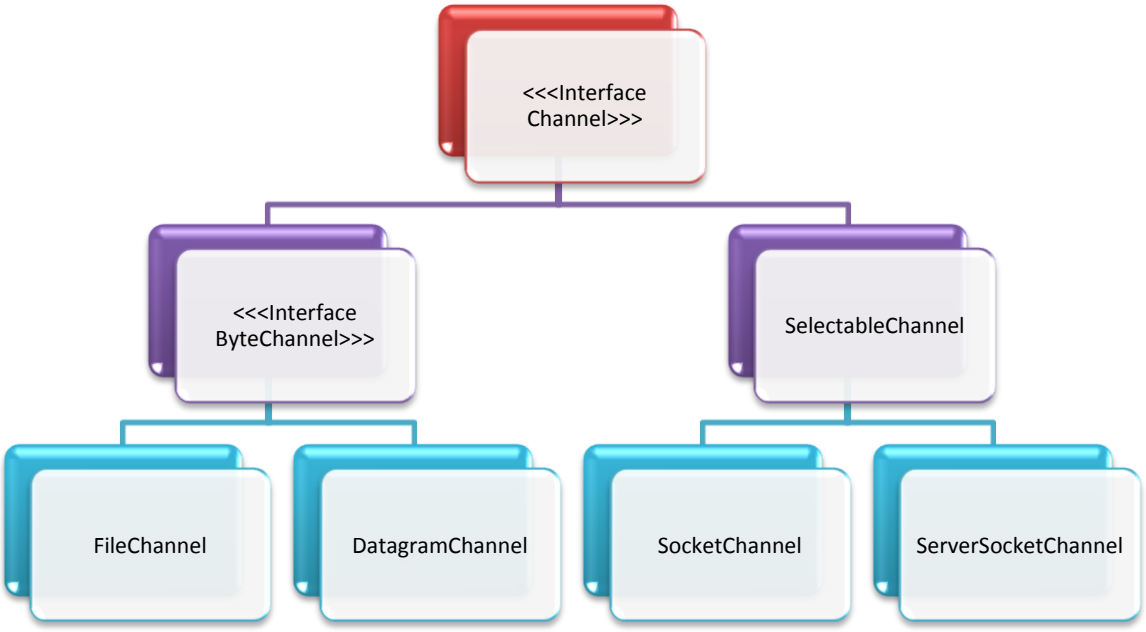


Figura 1.6. Jerarquía Simplificada de Java.nio.channel

1.5.3.1 java.nio.channels.ServerSocketChannel

La clase `java.nio.channels.ServerSocketChannel` es un canal seleccionable para sockets TCP pasivos. Por decirlo de manera llana, un objeto `ServerSocketChannel` viene a ser un envoltorio para un objeto `ServerSocket`, al cual asocia un canal. Para crear un objeto `ServerSocketChannel` se usa el método `public static ServerSocketChannel open() throws IOException.`

Un `ServerSocketChannel` recién creado no está ligado a una dirección ni a un puerto. La ligadura se consigue mediante uno de los métodos `bind()` de la clase `java.net.ServerSocket`.

El método `public abstract ServerSocket socket()` devuelve un socket de servidor asociado con el canal.

El método `abstract SocketChannel accept() throws IOException` acepta una conexión hecha al socket del canal. Si el `ServerSocketChannel` está en modo no bloqueado, devolverá `null` si no hay conexiones pendientes; en caso contrario, se bloqueará indefinidamente hasta que llegue una conexión o se produzca una excepción de E/S.

El método `public abstract SelectableChannel configureBlocking(boolean bloqueo) throws IOException, ClosedChannelException, IllegalBlockingModeException` procede de la clase `java.nio.channels.SelectableChannel` y establece si el canal podrá bloquearse o no. Es decir, determina si las operaciones de E/S sobre el canal, como `write()` o `read()`, lo bloquearán o no.

A diferencia de los otros canales, un `ServerSocketChannel` sólo puede establecer conexiones. La transmisión de datos se realiza mediante los canales asociados a los sockets que se obtienen de él cuando se llama al método `accept()`. Si se configura sin bloqueo un `ServerSocketChannel`, el método `accept()` no bloqueará el programa en ejecución hasta que reciba una conexión.

La clase `java.net.InetSocketAddress`, implementa una dirección IP de socket mediante un par (dirección IP, número de puerto) o (nombre del ordenador, número de puerto).

Si un socket se configura sin bloqueo, se cumple lo siguiente:

a) *Una llamada a* `read()` *transferirá los bytes disponibles en el momento de la llamada. Si no hay datos disponibles, devolverá* `0`*.*
b) *Una llamada a* `write()` *transferirá a un socket los datos disponibles en ese mismo momento. Si no hay datos disponibles, devolverá 0.*
c) *Una llamada a* `accept()` *devolverá* `null` *si en ese momento no hay ningún cliente que intente conectarse.*

1.5.3.2 java.nio.channels.SocketChannel

La clase `java.nio.channels.SocketChannel` es un canal seleccionable para sockets TCP activos. Un objeto `SocketChannel` viene a ser un envoltorio para un objeto `Socket`, que permite asociarle un canal. Las operaciones de enlazado con una dirección IP y un puerto, de cierre, etc., se realizan mediante el socket asociado.

Los objetos de la clase `SocketChannel` se crean mediante llamadas a los métodos estáticos `open()` de la clase. El método `public abstract boolean connect(SocketAddress direccionRemota) throws IOException` conecta un canal de socket con un objeto `InetSocketAddress`. El método `public abstract Socket socket()` devuelve el socket asociado al canal.

Al igual que sucede con los `ServerSocketChannels`, los `SocketChannels` también pueden configurarse sin bloqueo, de forma que las operaciones de E/S no bloqueen el programa en ejecución (ver figura 1.7).

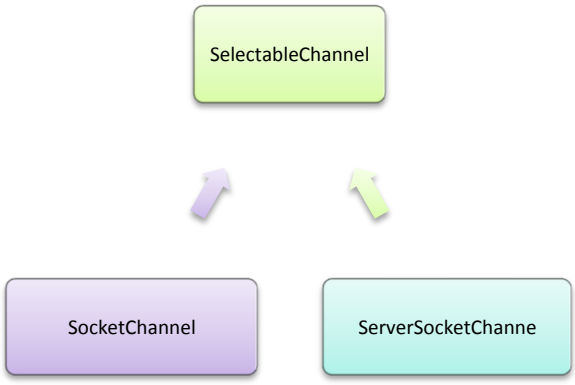


Figura 1.7. Clase SelectableChannel

1.5.3.2.1 java.nio.channels.Selector y java.nio.channels.SelectionKey

La clase `java.nio.channels.Selector` es una de las principales de la API NIO. Un objeto *Selector* controla una serie de canales y lanza un aviso cuando uno de ellos lanza un suceso de E/S. La clase `Selector` informa a la aplicación de las operaciones de E/S que ocurren en los canales que ésta mantiene activos.

La información sobre las operaciones de E/S se registra en un conjunto de claves, que son instancias de la clase `java.nio.channels.SelectionKey`. Cada clave almacena información sobre el canal que desencadena la operación y el tipo de ella (lectura, escritura, conexión entrante, conexión aceptada).

En la clase *Selector*, las instancias se crean con el método estático `public static Selector open() throws IOException.`

El método `public abstract int select() throws IOException` bloquea el programa hasta que algún canal recibe datos.

El método `select()` se bloquea hasta que uno o más canales reciban algún suceso de E/S. Con una llamada a `select()` se espera simultáneamente a todas las entradas de los clientes.

Para saber qué sucesos E/S de los que estamos interesados se producen en un determinado canal es necesario registrar el canal con el selector y especificar el tipo o los tipos de sucesos de interés. Esto se realiza mediante el método `public final SelectionKeyregister(Selector sel, int ops) throws ClosedChannelException` del canal.

Se dice que una clase es *seleccionable* si puede registrarse con un selector. Todos los canales que descienden de la clase `java.nio.channels.SelectableChannel` son seleccionables. Un canal seleccionable es un canal que puede multiplexarse.

Los sucesos de E/S para los que se puede registrar un canal mediante un selector se especifican con las siguientes constantes enteras:

- `SelectionKey.OP_READ.` Registra el canal para sucesos de lectura.
- `SelectionKey.OP_WRITE`. Lo registra para sucesos de escritura.

- `SelectionKey.OP_ACCEPT`. Lo registra para las peticiones entrantes de los clientes (se usa en servidores).
- `SelectionKey.OP_CONNECT`. Lo registra para las conexiones aceptadas por los servidores (se usa en clientes).

Distintos canales pueden registrarse, para diversos sucesos, en un mismo selector; de ahí que se diga que un conjunto de canales se multiplexa en un selector

Un canal `ServerSocketChannel` puede registrarse con `OP_ACCEPT, OP_READ` y `OP_WRITE`. Así mismo, uno `SocketChannel` puede registrarse con `OP_CONNECT, OP_READ` y `OP_WRITE`.

Cada objeto `SelectionKey` representa la clave o identificador de un canal que ha lanzado un suceso de E/S. Por ello, el conjunto de `SelectionKeys` que devuelve una llamada al método `selectedKeys()` es equivalente a un conjunto de canales.

Un `SelectionKey` es válido hasta que se cierra el canal, el selector o se llama al método `cancel()`.

Para eliminar un canal de un selector (desregistrarlo), se usa el método `public abstract void cancel()` del objeto `SelectionKey` asociado. Si se cancela una clave, su canal se desregistrará del selector durante la próxima operación con `select()`. Si un canal se cierra, se desregistra de todos los selectores donde esté registrado.

El método `public abstract void close() throws IOException` cierra el selector. Al cerrarse, hace lo siguiente: a) invalida todas las claves asociadas con él; b) desregistra todos los canales asocidos; y c) libera los recursos del sistema operativo que tenía ocupados.

El método `public abstract Set selectedKeys() throws ClosedSelectorException` devuelve un conjunto (`Set`) de claves correspondientes a los canales que han recibido operaciones de E/S. Cuando uno o más sucesos de E/S han sido lanzados por cualquiera de los canales registrados, puede accederse así al conjunto de claves de los canales con sucesos:

```
Set claves = selector.selectedKeys();
```

Los métodos de la clase:

- `SelectionKey public final boolean isAcceptable() throws CancelledKeyException,`
- `public final boolean isConnectable()throws CancelledKeyException,`
- `public final boolean isReadable()throws CancelledKeyException`
- y `public final boolean isWritable()throws CancelledKeyException`

comprueban el tipo de suceso producido en el canal seleccionado (conexión entrante, conexión aceptada, recepción de datos o envío de datos).

Es necesario usar el método `remove()` tras procesar un canal; si no, el suceso o los sucesos ya procesados volverían a lanzarse y a tratarse cuando se produjera cualquier otro nuevo suceso.

Básicamente el funcionamiento de la clase `Selector` es:

• En un selector se registran varios canales, cada uno para ciertos tipos de sucesos de E/S. Si un canal no se registra para un suceso, éste no será comprobado por el selector.
• El selector detecta los sucesos que se producen en cada canal y permite que la aplicación decida qué hacer en respuesta a cada suceso.

1.6 Sonido en Java

El sonido de Java requiere que los programadores conozcan como funcionan sus clases principales. Algunas de estas clases son `AudioSystem`, `AudioFormat` y `SourceDataLine`. `AudioSystem` está conectada a los dispositivos del sistema de forma que podemos acceder a ellos además de que dentro de ella tenemos métodos para tratar el `stream` de audio. `AudioFormat` con el que manejaremos el formato de audio que necesitaremos. Y `SourceDataLine` que nos proporciona la entrada al dispositivo de sonido, es decir, que podremos introducir nuestro sonido en el dispositivo y de esta forma escucharlo).

Java proporciona una clase más directa que es *CLIP*. Suele utilizarse más para sonidos pequeños en aplicaciones o en *applets*. Una de las limitaciones de *CLIP*, es que necesita cargar el sonido entero en memoria antes de reproducirlo lo cual aumenta los recursos utilizados, además de que sólo está preparada para reproducir los formatos típicos en *Java* (*Wav*, *A-Law*, *U-Law*, etc.).

Dado que vamos a usar una interfaz gráfica para la ejecución del sonido, es necesario programar Java con hilos (`Thread`), por lo que nuestro reproductor heredara de la clase `Thread` y en el momento de reproducir, se lanzará un hilo que será el encargado de reproducir el sonido mientras seguimos teniendo el control en nuestra aplicación. Un `Thread` es una secuencia de instrucciones que se ejecuta en paralelo con la aplicación que lo invoca, podríamos decir que mientras nosotros usamos nuestra aplicación, lanzamos otra pequeña aplicación que reproducirá nuestro sonido sin interferir en la primera (ejecución en paralelo).

Para realizar el proceso de reproducción de sonido, básicamente se hace uso de tres métodos.

• `Run()`. Desde este método es de donde comienza el proceso del tratamiento de nuestro archivo *MP3*. Desde aquí obtendremos el `InputStream` y el `AudioFormat` y una vez que los tengamos, pasaremos a llamar al siguiente método `Reproducir()`, que podria llamarse `cancionesrun()` ya que es una implementación del método `run()` que heredamos de la clase `Thread`. De esta forma al invocar el método `start()` del hilo nuevo, directamente ejecutara el método `run()`.

• `Reproducir()`. Aquí es donde creamos el buffer de *Bytes* y el `SourceDataLine`. De esta forma, recibimos el `AudioInputStream` (el `stream` de audio) y crearemos el `SourceDataLine` (que recibimos desde el método `getLine` que). Luego entraremos en un bucle en el que contaremos los bytes que leamos y los bytes que escribamos, y desde el que leeremos (llenando el buffer) desde el `AudioInputStream` y luego lo escribiremos en el dispositivo de sonido desde el `SourceDataLine`. Finalmente, cuando se termina la canción, cerraremos el `SourceDataLine` y el `AudioInputSource`.

• `GetLine()`. Desde este método creamos el `SourceDataLine` (desde `AudioSystem`) a partir del stream ya descodificado y nos lo devolverá. Para este caso, se usará un método que implementa el JDK1.5 que es el `getSourceDataLine()`.

Una forma más fácil de para reproducir sonido haciendo uso de los elementos que tenemos de Java, es haciendo uso de las librerías que hemos instalado de `Javazoom`, en `JLayer` concretamente (es la biblioteca para decodificar, reproducir y convertir MP3 para la plataforma Java). Existe una clase que crea un reproductor de *MP3* completo al que únicamente hemos de pasarle el `InputStream` (con `FileInputStream`) cuando lo creamos, y al ejecutar el método .`play()` de dicho reproductor y él se encargará del resto.

La clase `player` implementa una sencilla forma de reproducir audio MPEG. Dentro de sus principales métodos encontramos:

- `close()`. Se encarga de cerrar la reproducción. Toda la aplicación de audio, se detiene inmediatamente.
- `decodeFrame()`. Decodifica un sola trama
- `getPosition()`. Indica la posición en milisegundos de la actual muestra de audio que se está reproduciendo.
- `isComplete()`. Devuelve el estado de terminado en el reproductor.
- `play()`. Reproduce una serie de sonidos de audio MPEG.

Capítulo II

Justificación de Necesidades

En este capítulo se hablará de la causa por la cual se optó por el diseño de este proyecto y las justificaciones que confirman que es un proyecto viable. También hacemos mención del tipo de investigación que se utilizó para su desarrollo en comparación con otros diseños comerciales.

2.1 Tipos de investigación.

La Investigación es un proceso que, mediante la aplicación del método científico, procura obtener información relevante para extender, verificar, corregir o aplicar el conocimiento. Se divide en:

- Bibliográfica.
- De campo.
- Experimental.

Para nuestro caso, usamos el método bibliográfico para la recopilación de información a base de encuestas realizadas con anterioridad.

En un sentido amplio, el método de investigación bibliográfica es el sistema que se sigue para obtener información contenida en documentos. En sentido más específico, el método de investigación bibliográfica es el conjunto de técnicas y estrategias que se emplean para localizar, identificar y acceder a aquellos documentos que contienen la información pertinente para la investigación. Permite apoyar la investigación, conocer lo que se ha hecho para continuar la investigación y buscar información relevante y/o novedosa.

De acuerdo con las encuestas encontradas, se recopiló la siguiente información.

Para una encuesta realizada a una población de 102 personas para la siguiente pregunta, se encontraron los siguientes resultados de los diferentes mensajeros propuestos.

2.2 Resultados de investigación recopilada

¿Qué Mensajero Instantáneo Usas?

Windows Messenger	**7**	**6,86%**
MSN Messenger	**22**	21,57%
Yahoo! Messenger	**5**	4,90%
Messenger Plus Live	**9**	8,82%
Windows Live Messenger	**55**	53,92%
AOL Instant Messenger	**0**	0%
Skype	**0**	0%
Gaim	**1**	0,98%
Otro ...	**3**	2,94%

Tabla 2.1 Grafica de resultados obtenidos del uso de mensajeros comerciales

El 64% de los usuarios de Internet de América latina utiliza algún programa de mensajería instantánea, marcando una presentación de mercado superior a la que se da en Europa (49%) y EE.UU. (37%), de acuerdo a un estudio realizado por **comScore**, sobre la utilización de este tipo de programas de comunicación en línea[1].

[1] http://www.pilarteens.com.ar/modules.php?name=News&file=article&sid=15

Según el informe, en el viejo continente se registra el mayor número de usuarios conectados a servicios de mensajería instantánea, con 82 millones de usuarios, superando a Estados Unidos, que registra un total de 69 millones[2].

El MSN Messenger sigue siendo el mensajero instantáneo más utilizado, con una penetración a nivel mundial del 61%[3].

El 90% de los usuarios de América latina y más del 70% de Europa y Asia Pacífico utilizan el programa mensajería de Microsoft. Sin embargo, en Estados Unidos debe pelear el liderazgo con sus competidores más directos, el AIM de AOL y el Yahoo! Messenger, con quienes se reparte entre el 27% y 37% del mercado[4].

A nivel mundial, el 14% de los usuarios de mensajeros utiliza el software de comunicaciones de VoIP[5].

Asia Pacífico es quien más usuarios conectados a este servicio registra, con un 36% del total, frente al 3% que lo utilizan en Estados Unidos[6].

Más de la mitad de los 13 millones de cibernautas mexicanos utiliza cotidianamente el mensajero instantáneo (MI), ubicándose México como el tercer país en el mundo en el uso del MSN Messenger, después de Estados Unidos e Inglaterra, de acuerdo con datos de T1msn[7].

Un estudio reciente de la Asociación Mexicana de Internet (*AMIPCI)* señala que un 61 por ciento de los internautas mexicanos utiliza algún programa de mensajería, mientras que en Estados Unidos, un estudio de *Pew Internet* & *American Life Project* indica que el 42 por ciento de los usuarios de internet reportó usar mensajeros instantáneos. Sin embargo, la cifra de la AMIPCI sobre México difiere con la del medidor de audiencias *MediaMetrix* de *Comscore*, que asegura que 9 millones de mexicanos utilizan el mensajero instantáneo, lo que se traduce en un 69 por ciento[8].

El uso del chat se ha vuelto tan popular que 2 mil 500 millones de mensajes se envían cada día en el mundo tan sólo por MSN Messenger, el mensajero de Microsoft. Si los usuarios de este servicio de mensajería formaran un país, su población sería idéntica a la de Japón, el décimo país más poblado del mundo[9].

Las estadísticas en México para el uso de mensajeros instantáneos, arrojaron los siguientes resultados mostrados en las figura 2.1:

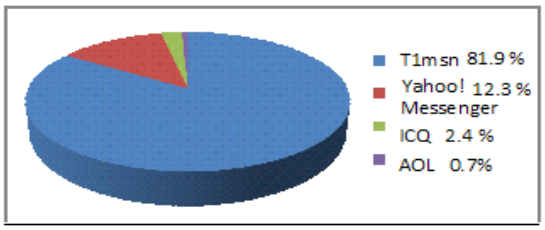


Figura 2.1 Grafica de resultados obtenidos del uso de mensajeros comerciales en México

[2, 3, 4, 5, 6] http://www.pilarteens.com.ar/modules.php?name=News&file=article&sid=15

[7, 8, 9] http://pcdoctor.com.mx/Radio%20Formula/temas/Messengers.htm

77 por ciento de los usuarios entre 18 y 25 años de edad utiliza un mensajero instantáneo, comparado con 70 por ciento de usuarios entre los 25 y 34 años.

Todo ello, con la finalidad de ver que tan viable es el mensajero instantaneo en un ámbito de aprendizaje virtual y asi poder conocer las necesidades de los usuarios.

Dado que para las estadísticas el mensajero más usado es el Windows Live Messenger, suponemos que no es recomendable para nuestros objetivos el hacer uso de él ya que contiene diferentes tipos de distracciones como es el envío y recepción de imágenes, el servicio de cámara web y por lo tanto establecer video llamadas.

Los mensajeros instantáneos comerciales entre sus funcionalidades incluyen juegos, lo que ocasiona la perdida de atención a los usuarios y por los tanto no permiten que los usuarios puedan realizar actividades de estudios.

Otra desventaja es que al hacer un envío y recepción de archivos a través de la red, puede ocasionar que se transfieran virus y que infecten a nuestros equipos sin restricción alguna, (ver tabla 2.2).

Los resultados obtenidos mediante estas encuestas, nos ayudan a comprender el enfoque que se le debe de dar al mensajero, así mismo la utilidad que pudiera tener ahora y en un futuro como una herramienta de apoyo para los estudiantes y profesores, ya que cuenta con seguridad en cuanto a los virus y que el uso de esta herramienta será de uso exclusivo para la comunidad de la ESIME Zacatenco.

Otra ventaja del mensajero es que cuenta con funcionalidades que no permitan que los estudiantes se distraigan y que puedan mantenerse enfocados en sus actividades.

De forma general, suponemos que el empleo de este proyecto en la comunidad estudiantil, será de gran ayuda y que es posible que más del 90% de los estudiantes puedan hacer uso de él, ya que en la escuela se cuentan con servicios de cómputo de fácil acceso y por lo tanto, cualquier estudiante puede tener acceso a él.

En la tabla 2.2 se muestran las funciones de este proyecto en comparación con los mensajeros comerciales que se encuentran en el mercado.

Funcionalidad	Windows Messenger	MSN Messenger	Yahoo! Messenger	Messenger Plus Live	Windows Live Messenger	Mensajero Instantáneo
Envió y Recepción de imágenes		✓	✓	✓	✓	
Servicio de cámara Web		✓	✓	✓	✓	
Uso de videojuegos		✓	✓	✓	✓	
Servicio de videoconferencia	✓	✓		✓	✓	
Envió y recepción de archivos	✓	✓	✓	✓	✓	
Envió y recepción de menajes de texto	✓	✓	✓	✓	✓	✓

[10] http://pcdoctor.com.mx/Radio%20Formula/temas/Messengers.htm

Capítulo III

Desarrollo de la Aplicación

En este capítulo se presenta de manera detallada la forma en la que se desarrolló el proyecto en cada una de sus etapas y las herramientas que se utilizaron para su diseño, además de su descripción y funcionamiento.

3.1 JFC (SWING)

Java proporciona una biblioteca de clases denominadas JFC *(Java Foundation Classes*) y actualmente bajo esta denominación se agrupan las siguientes interfaces para programación de aplicaciones:

- AWT *(Abstarct Window Toolkit)*. Conjunto de componentes para diseño de interfaces gráficas de usuario.
- Swing. Conjunto de componentes para diseño de interfaces gráficas de usuario, derivadas de AWT.
- Accesibilidad. Ofrece soporte para usuarios con limitaciones.
- Java 2D. Ofrece gráficos de 2D de alta calidad, texto e imágenes.
- Soporte para arrastrar y colocar *(drag and put)*.

Actualmente, Swing ha llegado a desplazar a AWT porque ofrece un conjunto de componentes escritos en Java con una mayor funcionalidad, y trabaja de manera independiente de la plataforma.

La principal diferencia entre AWT y Swing es que cada componente de AWT tiene asociado su propio recurso nativo de ventana. Esto quiere decir que tiene dependencia de la plataforma esto por implementar código nativo, por otro lado, este tiene un gran consumo de recursos en los programas que incluyen un gran número de componentes. Por esta razón, a estos componentes se les atribuyó el calificativo de "pesados". A diferencia los componentes creados en Swing, son creados con Java, lo que les hace independiente de las plataformas y no tiene su propia ventana; son sencillamente dibujados como imágenes sobre sus contenedores, esto les proporciona menor consumo de recursos razón por lo que se les da el calificativo de "ligeros".

3.1.1 Componentes Swing.

Todos los componentes Swing son objetos de clases derivadas de la clase **`JComponet`**, que a su vez se deriva de la clase **`java.awt.Component`**, lo que pone de manifestó que Swing se deriva de AWT. Swing provee los contenedores de nivel alto **`JWindow, JFrame, JDialog`** y **`JApplet:`**

- `Java.lang.Objet`
  - `Java.awt.Component`
    - `Java.awt.Container`
      - `Javax.Swing.`**`JComponent`**
      - `Java.awt.Panel`
        - `Java.applet.Applet`
          - `Javax.swing.`**`JApple`**
      - `Java.awt.Window`
        - `Java.awt.Dialog`

        - Java.swing.**JDialog**
    - Java.awt.Frame
        - Javax.swing.**JFrame**
    - Java.swing.**JWindow**

JWindow es una ventana sin barra de titulo y sin los botones que permiten su manipulación, que puede visualizarse en cualquier sitio del escritorio.

JFrame es una ventana con barra de titulo y con los botones que permiten su manipulación, que puede visualizarse en cualquier sitio del escritorio.

JDialog permite visualizar una caja de dialogo.

JApplet permite crear un applet swing; programa que visualiza una interfaz gráfica en el contexto de una página web.

Ya que Swing se deriva de AWT es posible que se mezclen en el mismo programa, aunque generalmente no se recomienda, a continuación se mencionan algunas diferencias:

- Todos los componentes AWT son pesados, y todos los componentes Swing son ligeros.
- Un componente ligero puede tener pixeles transparentes y además, no tienen porque ser rectangulares. Un componente pesado es opaco y siempre es rectangular.
- Cuando un componente ligero se solapa con un componente pesado, el componente pesado aparecerá enzima.

3.1.2 Manejadores de eventos swing

Java 2 proporciona varios manejadores de eventos, cada uno de los cuales maneja un tipo peculiar de eventos.

Los manejadores de eventos que manipulan los tipos de eventos serán implementados a partir de las interfaces que se exponen a continuación:

- **ComponentListener.** Permite manejar los eventos de tipo `ComponentEvent` generados por los componentes cuando cambian su tamaño, posición o visibilidad.
- **Focuslistener.** Permite manejar los eventos de tipo `FocusEvent` generados por los componentes cuando ganan o pierden el foco. Cuando un componente gana el foco está en condiciones de recibir entradas desde el teclado.
- **KeyListener.** Permite manejar los eventos de tipo `KeyEvent` generados por el componente que tiene el foco, cuando recibe entradas desde el teclado.
- **MouseListener.** Permite manejar los eventos de tipo `MouseEvent` generados por los componentes cuando el cursor del ratón entra en su área o sale de ella, y cuando el usuario pulsa y suelta el botón del ratón.
- **MouseMotionListener.** Permite manejar los eventos de tipo `MouseEvent` generados por un componente cuando el ratón se mueve sobre él.

3.1.3 Jerarquía de componentes de una aplicación

Los objetos fundamentales que intervienen en una aplicación en ejecución son: la ventana principal o contenedor del nivel superior, un contenedor de nivel intermedio y los componentes atómicos u objetos no destinados a contener a otros componentes.

Para obtener de forma automática la jerarquía de componentes de cualquier aplicación, la ejecutamos y pulsamos las teclas *Ctrl+Mayús+F1*. De esta manera podemos obtener la jerarquía de nuestra aplicación.

3.1.4 Cajas de texto, etiquetas y botones

Los componentes más comunes en una aplicación swing son las cajas de texto, componentes JTextField o JTextArea, son particularmente importantes porque permiten realizar la entrada de datos para una aplicación y visualizar el resultado producidos por la misma. Las etiquetas, componentes JLabel, son cajas de texto no modificables por el usuario. Su finalidad es informar al usuario de que tiene que hacer y cuál es la función de cada componente. Un botón de pulsación, componente de la clase JButton permite al usuario ejecutar una acción cuando sea preciso.

3.2 Diseño de la Interfaz Gráfica

Como ejemplo (ver figura 3.1), vamos a crear una aplicación que presente una interfaz gráfica con un mensaje de bienvenida. Para crear una aplicación de este tipo, básicamente siguiendo los siguientes pasos:

1. Crear un contenedor basado en una plantilla y colocarlo en un proyecto.
2. Añadir los componentes al contenedor.
3. Añadir los componentes al contenedor, editar sus propiedades y crear entre ellos las conexiones necesarias.
4. Editar el código fuente.
5. Compilar y ejecutar la aplicación.

Figura 3.1 Ventana de mensaje de bienvenida

Las operaciones de diseño de la interfaz gráfica, edición, compilación y ejecución de una aplicación, son más fáciles cuando se trabaja desde un entorno de desarrollo que integre todas esas operaciones. En este caso, haremos uso de NetBeans de Sun Microsystems.

Sun nos ofrece la posibilidad de utilizar este paquete sin ningún costo. NetBeans es un entorno de desarrollo intuitivo, personalizable, modular y extensible. Es un entorno integrado de desarrollo escrito en Java que agrupa un conjunto de utilidades para facilitar la edición, compilación, análisis y ejecución de cualquier programa Java. Además permite el diseño de interfaces gráficas de usuario sin escribir nada de código; basta con seleccionar uno a uno los componentes y colocarlos sobre una ventana, también facilita la creación de conexiones entre componentes, como por ejemplo, una conexión entre una caja de diálogo y un botón con el fin de abrir dicha caja cuando se pulse el botón. Cada una de estas acciones (añadir o eliminar un componente, establecer la conexión entre dos componentes, establecer una conexión entre dos componentes, etc.) se reflejará automáticamente en el código fuente.

Cuando se inicia NetBeans, el espacio de trabajo que se muestra en el mismo que mostraba cuando se cerró por última vez. Esto es, NetBeans cuando se cierra, guarda el estado actual de todos los espacios de trabajo. Los estados de trabajo a los que nos referimos, se corresponden cada uno de ellos con uno de las siguientes ventanas o paneles: explorador (incluye el sistema de archivos, el panel de proyecto por omisión y el panel de ejecución), edición (panel que muestra el código de la aplicación), edición de la GUI (incluye el editor de formularios, las paletas de componentes, el supervisor de componentes y la ventana de propiedades). La ventana de salida en la que se puede ver el resultado de la compilación o de la ejecución y la ventana de ejecución (Ver figura 3.2).

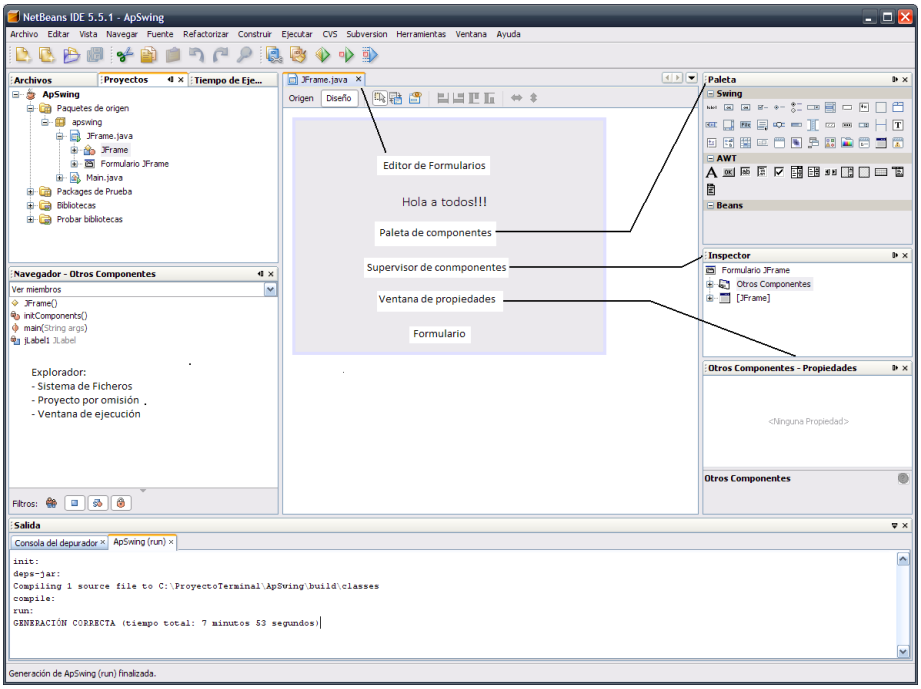


Figura 3.2 Barra de herramientas

El panel Filesystem del explorador muestra jerárquicamente los paquetes, objetos y archiveros que forman parte de la aplicación. El supervisor permite ver los componentes que forman la interfaz gráfica. La ventana de propiedades permite editar las propiedades del componente seleccionado (tamaño, color, fuente, etc.), los manejadores de eventos, etc.

El editor de código permite editar el código fuente y proporciona ayuda para completarlo mientras se está escribiendo. Y el editor de formularios permite colocar los componentes en un formulario.

3.2.1 Crear un contenedor basado en una plantilla.

Los contenedores son componentes Swing utilizados para ubicar otros componentes. Por ejemplo, en la figura 3.3, correspondiente a la interfaz gráfica de la aplicación ApSwing que queremos desarrollar, explica como se acomodan los componentes en uno de estos contenedores.

Para agregar componentes a una ventana, es aconsejable utilizar un contenedor intermedio; esto facilitará la realización de otras operaciones posteriores como, por ejemplo, añadir un borde alrededor de los componentes. Siempre que sea necesario, un contenedor puede contener a otros contenedores, lo que dará lugar a una jerarquía de contenedores que facilitará la distribución de los componentes. La raíz de esa jerarquía es el contenedor del nivel superior definido por el marco de la ventana.

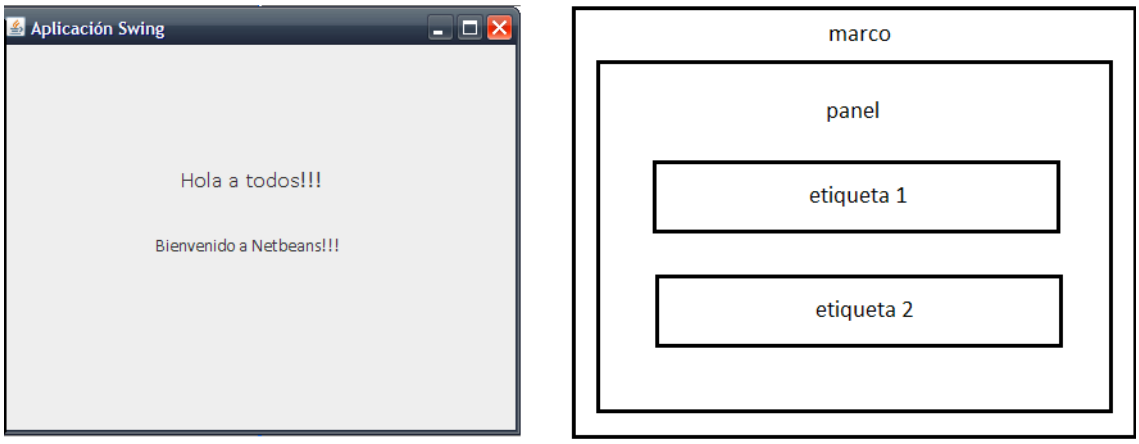


Figura 3.3 Jerarquía del contenedor

Como ejemplo, analizaremos la figura 3.3. En ella se puede observar:

- Un marco (objeto de la clase `JFrame`) que se corresponde con la ventana marco principal. Es el contendor del nivel superior en la jerarquía. Los contenedores correspondientes a `JDialog` y `JApplet` son también del nivel superior. Cada contenedor del nivel superior tiene de forma predeterminada un panel raíz al que se puede acceder por medio del método `getContentPane`. Se trata de un panel de contenido que permite ubicar otros componentes (paneles y controles). Por ejemplo, la siguiente sentencia añade al objeto denominado panel al panel raíz determinado:

```
getContentPane().add(panel.BorderLayout.CENTER);
```

- Un panel (objeto de la clase `JPanel`) que corresponde con el área de trabajo. Es el contenedor de un nivel intermedio llamado panel de contenido. Su propósito es simplificar la colocación de controles. Un panel de contenido puede incluir a su vez otros paneles de contenido. La siguiente sentencia crea un panel de este tipo:

```
Jpanel panel = new Jpanel();
```

- Dos etiquetas (objetos de la clase `Jlabel`). Este tipo de componentes es lo que llamamos controles. Cada uno de ellos realiza una operación específica. Para añadirlos a un panel determinado, se utiliza el método add. Por ejemplo:

```
panel.add(etiqueta1);

panel.add(etiqueta2)
```

Después de esta introducción teórica, pasemos a la parte práctica. El siguiente paso, después de seleccionar su sistema de archivos en el panel *Filesystems,* es elegir la plantilla que proporcionará el contenedor del nivel superior. Para ello, ejecute la orden *New…* del menú *File*. Se visualizara una ventana mostrando las plantillas disponibles. *NetBeans* proporciona plantillas para *applets*, clases, cajas de diálogo, *beans,* páginas Web, etc., reduciendo de esta forma el tiempo y esfuerzo necesario para escribir un programa. Una plantilla determina la apariencia y la conducta inicial del objeto que se crea partir de ella.

Como ejemplo, seleccione la plantilla **`jFrame`** del nodo *Java GUI Forms;* en la caja inferior puede observar una breve descripción acerca de la plantilla seleccionada. Pulse el botón *Next* para continuar con la selección de la ubicación de la aplicación.

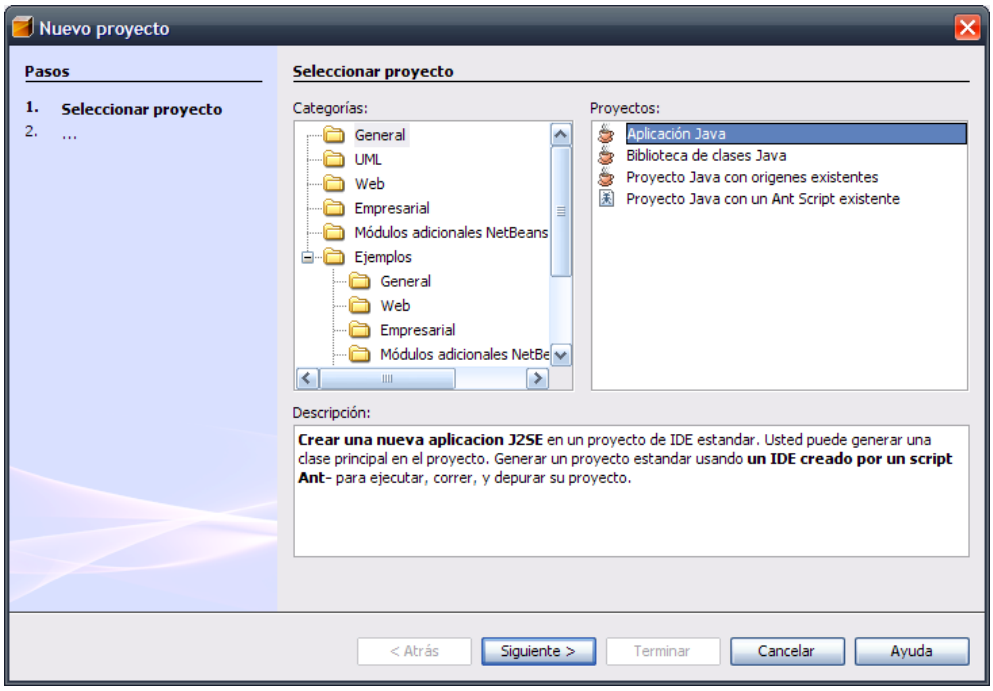


Figura 3.4 Plantilla JFrame

Este diálogo le permite escribir el nombre de la clase aplicación en la caja de texto *Name* (en nuestro caso *ApSwing*), seleccionar en el sistema de archivos la carpeta donde se va a guardar la aplicación, o bien, si la carpeta no existe, escribir en la caja de texto *Folder* el nombre de la carpeta donde se guardaran los archivos de la aplicación. Haga clic en el botón *Create* para crearla (puede observar que la ruta de esta carpeta se muestra en la caja *Directory; Folder* define el paquete donde se ubicará la clase). Finalmente, haga clic en el botón *Finish* (ver figura 3.5).

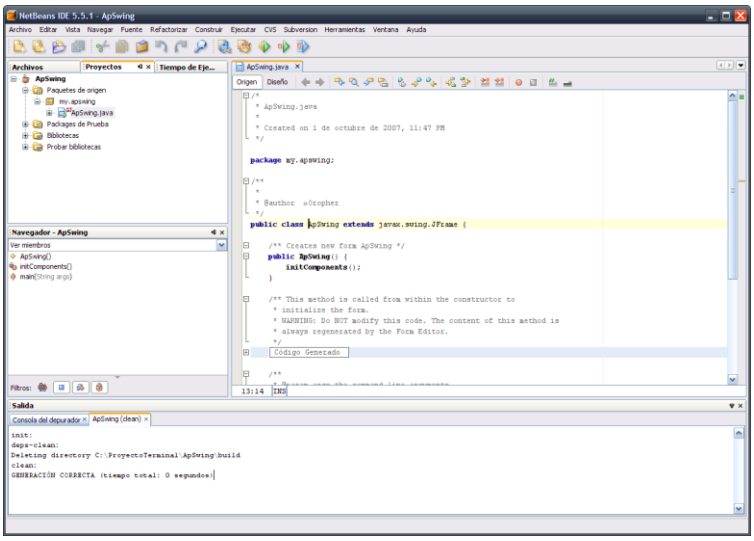


Figura 3.5 Ventana de edición de código

En la ventana de edición de código se puede observar que se ha generado una clase *ApSwing* perteneciente al paquete *Cap01.AsPwing,* con un constructor público y una serie de métodos. En los pasos siguientes añadiremos los componentes al formulario (pestaña *ApSwing [Form]*), utilizando el editor de formularios y el código necesario (pestaña *ApSwing*) para que la aplicación realice lo deseado.

Para añadir los componentes al formulario, primero seleccione el panel *ApSwing [Form],* y después, haciendo clic con el botón derecho del ratón sobre el formulario, se mostrara un menú contextual que nos permitirá realizar muchas operaciones, entre ellas, elegir el componente *swing* que deseamos añadir a la interfaz gráfica que estamos diseñado.

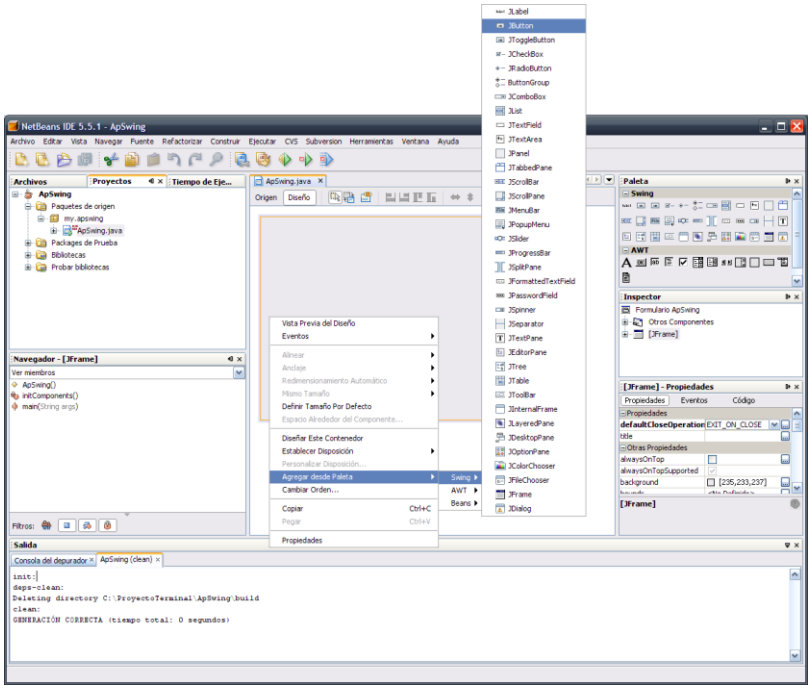


Figura 3.6 Componentes del formulario

Para editar el código necesario para que la aplicación realice las tareas deseadas, primero seleccione al panel *ApSwing,* y después, haciendo clic con el botón derecho del ratón sobre la ventana de edición, se mostrara un menú contextual que nos permitirá realizar muchas operaciones, entre ellas, añadir una nueva clase, un nuevo método, un nuevo atributo, etc. Dependiendo del lugar donde haga clic en la ventana de edición, las opciones será diferentes. También, puede optar por realizar esta tarea manualmente.

3.2.2 Ejecutar la Aplicación

Si ahora compilamos y ejecutamos esta aplicación, para lo cual tendremos que elegir la orden *Execute* del menú *Build* (F6) o bien hacer clic en el botón correspondiente de la barra de herramientas de la ventana principal, aparecerá sobre la pantalla la ventana de la figura mostrada a continuación (aparecerá reducida a la barra de título, ya que no tiene componentes) y podremos actuar sobre cualquiera de sus controles (minimizar, maximizar, mover, ajustar el tamaño, etc.).

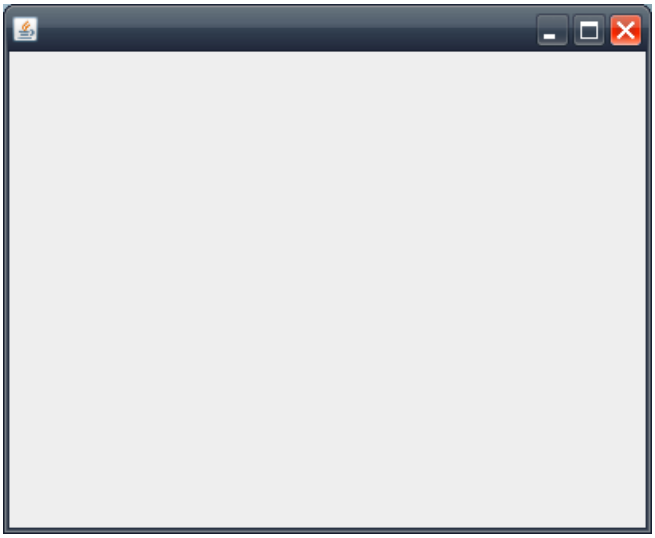


Figura 3.7 Ventana de aplicación

Para finalizar, haga clic en el botón  para cerrar la ventana.

3.2.3 Editar el código fuente

Supongamos ahora que deseamos añadir un título a la ventana y fijar su tamaño inicial. Una forma de hacer esto es proporcionar tanto el título como el tamaño en el momento de crear el objeto **`JFrame`.** Esto quiere decir que los métodos del objeto que permitan estas propiedades deben ser invocados desde el constructor *ApSwing.* El método que permite establecer el título es **setTitle** y el que permite establecer el tamaño es **`setSize`.**

Una forma rápida de situarse en el editor sobre un determinado método para modificarlo es localizarlo en el *explorador* y hacer doble clic sobre él. Observando la figura siguiente, podrá comprobar que haciendo doble clic sobre el constructor *ApSwing* será mostrando el editor de código fuente situado en el punto de inserción sobre este método.

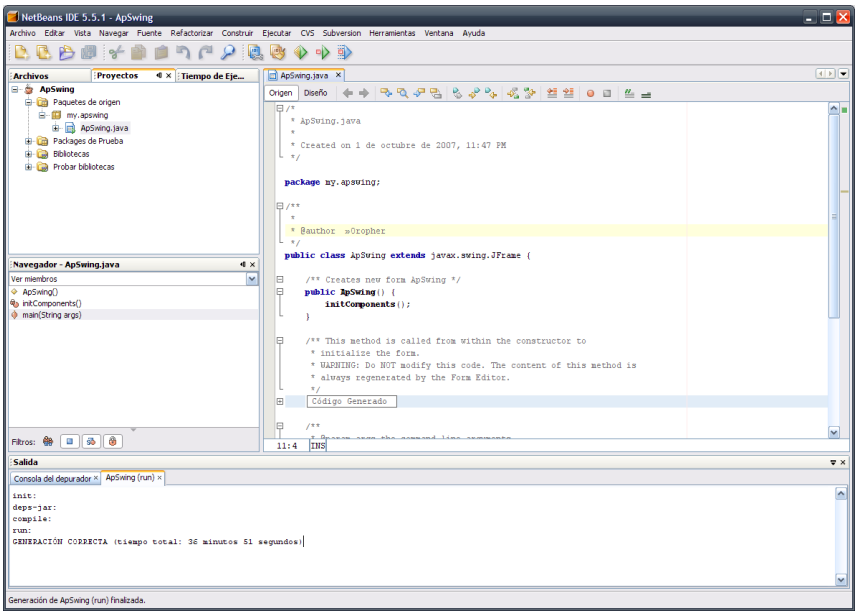


Figura 3.8 Ventana de edición

Observe que este método invoca a initComponents para ejecutar las operaciones de iniciación requeridas para los componentes de la aplicación. Después hemos añadido dos líneas: una para establecer el título de la ventana y otra para establecer su tamaño. Observe también que initComponents había ejecutado el método pack para ajustar el tamaño de la ventana al mínimo que permita visualizar todos sus componentes. Esta característica queda ahora anulada por setSize.

3.2.4 Añadir los componentes al contenedor

Los componentes, tales como cajas de texto, botones, etiquetas, marcos, listas y temporizadores, son objetos gráficos que permiten introducir o extraer datos. El contenedor más los componentes dan lugar al formulario que hace de interfaz o medio de comunicación con el usuario.

Para añadir un componente a un contenedor, primero mostraremos el editor de formularios. Para ello, seleccione el editor de formularios haciendo clic en la pestaña *ApSwing [Form]* o haga doble clic sobre el nodo *JFrame* en el explorador. Después nos dirigiremos a la paleta de componentes para seleccionar lo deseado. La figura 3.9 muestra la paleta de componentes *Swing* de java proporcionada por *NetBeans:*

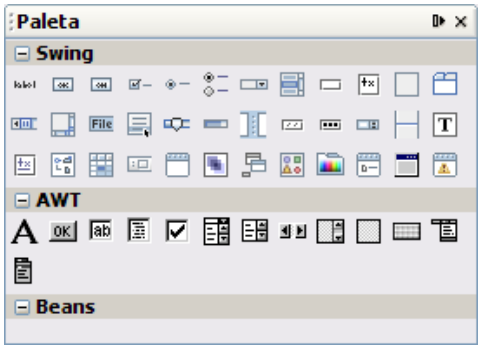


Figura 3.9 Paleta de componentes

Cada elemento de la paleta crea un único componente. Para saber de qué componente se trata, mueva el ratón encima del componente y espere a que se muestre la descripción asociada. Como se puede observar, estos componentes están clasificados en los siguientes grupos:

- ***Swing***. Grupo de componentes de interfaz gráfica de usuario independientes de la plataforma por estar escritos en java. Estos componentes ofrecen más y mejores funciones que los *AWT.*

- ***AWT***. Grupo de componentes de interfaz gráfica de usuario (GUI) que se han implementado utilizando versiones dependientes de la plataforma, sustituido en gran medida por el grupo de componentes Swing.

- ***Layouts***. Administradores de diseño para organizar los componentes que se añaden a un contenedor.

- ***Beans***. Componentes software reutilizables en cualquier programa. Mientras que los componentes anteriores son intrínsecos a Java, estos otros existen como archivos independientes con extensión *jar*. Se trata de programas Java que nosotros mismos podemos crear con una funcionalidad específica, con la intención de insertarlos posteriormente en otros programas.

3.2.5 Dibujar los Componentes

Añadir uno o más de estos componentes a un contenedor implica dos operaciones: seleccionar un administrador de diseño para dicho contenedor y dibujar sobre él los componentes requeridos.

Por ejemplo, volviendo a la aplicación que estamos desarrollando, diríjase al supervisor de componentes (Inspector) del editor de formularios y observe que el contenedor del formulario (JFrame) tiene asociado un administrador de diseño de tipo BorderLayout, ver figura 3.10:

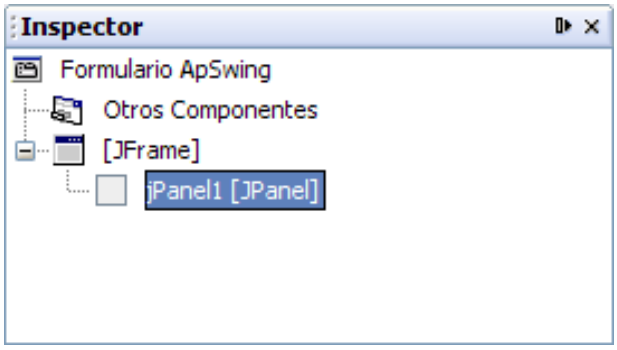


Figura 3.10 Supervisor de componentes

El administrador de diseño `BorderLayout` se caracteriza porque divide el contenedor en cinco secciones: norte, sur, este, oeste y centro, según muestra la figura 3.11.

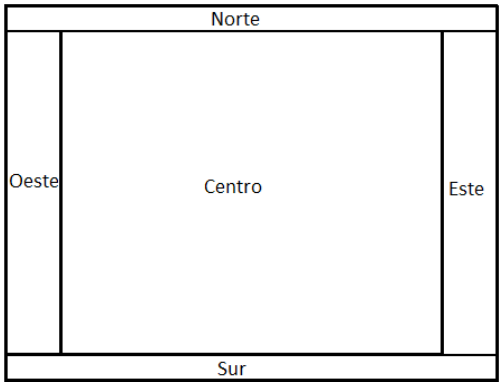


Figura 3.11 Administrador de diseño

Siguiendo el con consejo dado anteriormente al hablar de contenedores, vamos a colocar en el centro un panel al que asignaremos un borde que determinará los componentes que coloquemos sobre el mismo. Para ello:

1.- Haga clic en la pestaña Swing de la paleta de componentes.

2.- Haga clic en el componente Panel y después haga clic en el botón encontrado a su derecha para modificar los valores de esta propiedad. Se muestra la ventana siguiente:

3.- Con el panel seleccionado, haga clic en la propiedad *border* de la ventana de propiedades. Después, haga clic en el botón mostrado a su derecha para modificar los valores de esta propiedad. Se muestra la ventana de la figura 3.12:

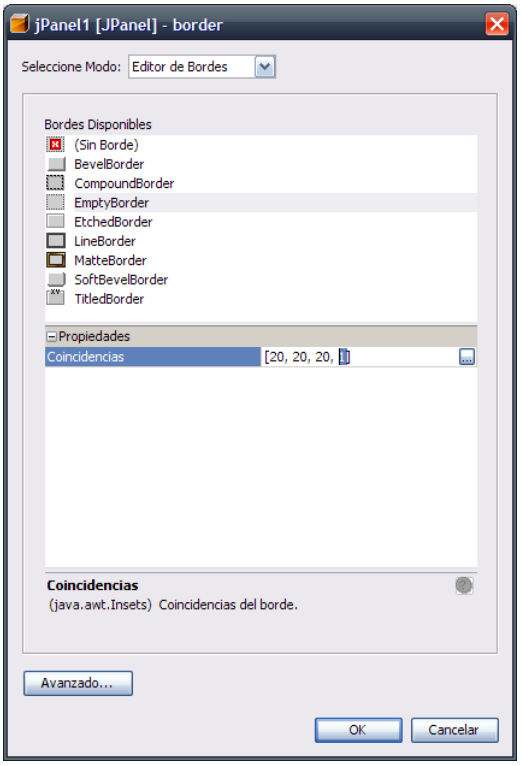


Figura 3.12 Ventana de propiedades

4.- Vamos a añadir al panel un borde sin pintar de ancho 20. Para ello, haga clic en el borde `EmptyBorder`, después en la propiedad `insets` y cambie los valores 1 por 20; pulse la entrada Entrar y después haga clic en OK.

De forma resumida vemos que se ha creado un objeto `jPalel1`, se le ha puesto un borde de 20 y se ha añadido al contenedor de nivel superior `JFrame`. Si quisiéramos cambiar el nombre del componente, se hace en el supervisor de componentes (*Inspector*), ver figura 3.13.

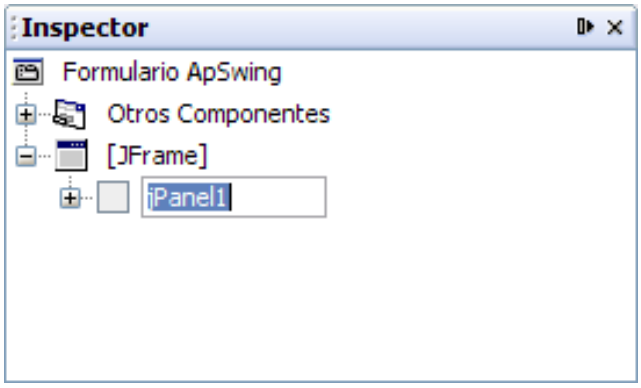


Figura 3.13 Ventana del supervisor de componentes

3.2.6 Organizar los componentes en un contenedor

Para organizar los componentes que se añaden en un contenedor, java proporciona los *administradores de diseño*. Un administrador de diseño determina el tamaño y la posición de los componentes dentro del contenedor. De forma predeterminada cada contenedor tiene asignado un administrador de diseño.

Swing proporciona seis administradores de diseño: **`FlowLayount`, `GridBaglayout`, `BorderLayout`, `CardLayout`, `GridLayout`** y **`Boxlayou`.**

Cuando queramos asignar a un contenedor un administrador de diseño diferente del predeterminado, primero hay que crearlo y después asignárselo invocando a su método **`setLayout`.** Por ejemplo, la siguiente sentencia asigna al contenedor *panel* un administrador de diseño de tipo **`GridLayout`** con 0 filas y 1 columna (los controles se colocarán en columna).

```
Panel.setLayout(new GridLayout(0,1));
```

Volviendo a nuestra aplicación, vamos a cambiar el administrador de diseño predeterminado del panel intermedio que acabamos de añadir, por un administrador de diseño de tipo GridLayout. Para ello, realice los pasos siguientes:

1. Haga clic en la pestaña *`Layouts`* de la paleta de componentes.
2. Haga chic en el componente *`GridLayout`* y después diríjase al formulario *`ApSwing`* y haga clic sobre el panel.
3. Vamos a modificar el número de filas y columnas de este administrador de diseño. Diríjase al supervisor de componentes y seleccione el componente *`GridLayout`* asociado con el panel para visualizar sus propiedades.
4. Cambie el número de columnas a 1 (propiedad *Columns)* y el número de filas a2 (propiedad *Rows).*

3.2.7 Añadir una etiqueta y editar sus propiedades.

Para añadir una etiqueta en el contenedor *jpanel1,* siga estos pasos:

1. Haga clic en la pestaña *Swing* de la paleta de componentes.
2. Haga clic el componente *jLabel* y después diríjase al formulario y haga clic en cualquier parte del panel. Observe que la etiqueta se coloca automáticamente en la primera sección libre del administrador de diseño.
3. Haga que el texto de la etiqueta aparezca centrado, en negrata y en tamaño 18. Para ello, haga clic en el formulario sobre la etiqueta, o bien diríjase al supervisor de componentes y seleccione el componente `jLabel` perteneciente al panel *jPanel1,* para visualizar sus propiedades.
4. Cambie el valor de la propiedad `horizontalAlignment` a `CENTER`. Observará en el formulario que ahora la etiqueta muestra el texto centrado.
5. Cambie el valor de la propiedad *font.* Para ello una vez seleccionada la propiedad , haga clic en el botón que hay a la derecha del valor que tiene asignado actualmente para visualizar el diálogo que la permitirá elegir el tipo de fuente, así como su estilo y tamaño.

Para que esta etiqueta muestre durante la ejecución el mensaje1, añada al constructor *ApSwing* la línea de código siguiente:

```
jLabel1.setText(obtener Mensaje1());
```

3.2.8 Añadir otra etiqueta

Para añadir otra etiqueta, puede repetir los pasos descritos en el apartado anterior, o bien realizar los siguientes:

1. Haga clic en la etiqueta que ha puesto sobre el formulario utilizando el botón derecho del ratón, y manteniéndolo pulsando ejecute la orden *Copy* (copiar).
2. Haga clic en cualquier parte del panel utilizando el botón derecho del ratón y ejecutando la orden *Paste* (pegar) del menú contextual que se visualiza. Observe que la etiqueta se coloca automáticamente en la primera sección libre del administrador de diseño.

3.2.9 Compilar la aplicación

Una vez finalizada la aplicación, puede compilarla y ejecutarla. El menú *Build* ofrece las siguientes posibilidades para compilar la aplicación:

- *Compile.* Compila el objeto seleccionado. Si el objeto es una carpeta, compila todas las clases que haya en la carpeta que aún no hayan sido compiladas o que hayan sido modificadas.
- *Compile All.* Igual que *Compile*, pero extendiendo la compilación a todas las subcarpetas de la carpeta seleccionada.
- *Build.* Compila todas las clases que componen la aplicación, independientemente de que hayan sido compiladas o de que hayan sido o no modificadas.
- *Build All.* Igual que *Build*, pero extendiendo la compilación a todas las subcarpetas de la carpeta seleccionada.

Si la construcción del archivo *.class* resulta satisfactoria, verá en la ventana de salida del compilador el mensaje "*Finished ApSwing*". Si hay problemas con la construcción, verá los mensajes de error correspondientes en la misma ventana.

Por ejemplo, suponga que en la primera línea de código del método *ApSwing* olvidó el punto y coma final. Cuando ejecute el comando *Compile, aparecerá una ventana mostrada en la figura 3.14*.

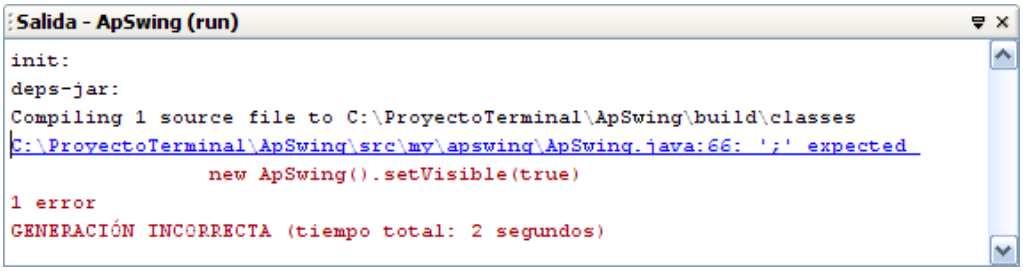


Figura 3.14 Ventana de compilación

La ventana de la figura anterior indica que el compilador ha detectado que falta un punto y coma. Para ir a la línea de código donde el compilador ha detectado el error y corregirlo, puede hacer doble clic sobre el mensaje de error. Una vez que obtenga una compilación sin errores, puede ejecutar la aplicación.

Para ejecutar la aplicación, seleccione la orden *Execute* del menú *Build*. Si no hay errores de ejecución, *NetBeans* mostrara los resultados.

No obstante, cuando la solución obtenida no sea satisfactoria y no seamos capaces de localizar dónde se está produciendo el error (imaginemos una expresión *al(2*b)* en la que olvidamos poner los paréntesis) podemos utilizar el depurador para ejecutar el programa paso a paso y poder investigar los resultados parciales que se van produciendo a medida que avanza la ejecución.

3.2.10 Proyectos

Un proyecto permite agrupar los archivos requeridos para producir una aplicación. Esto presenta ventajas como poder compilar todo el proyecto sin tener especificar los archivos que incluye, especificar la clase principal del proyecto, etc. De esto se deduce que para un determinado proyecto podemos configurar un escenario particular que será guardado cuando se finalice la sesión, lo que permitirá recuperarlo automáticamente la próxima vez que se cargue ese proyecto.

Para crear un nuevo proyecto hay que seguir los siguientes pasos:

1. Ejecute la orden `Project Manager` del menú `Proyect`.
2. Pulse el botón `New`, escriba el nombre que desea dar al proyecto en la ventana que se visualiza, y pulse el botón OK.
3. El siguiente paso es añadir el sistema de archivos donde se crea el nuevo proyecto.
4. Finalmente, cree la clase o clases que formarán la aplicación utilizando las plantillas adecuadas (`New`… del menú `File`) y guárdelas en la carpeta que desee.

El comando `Options` del menú `Tools` mostrado en la figura 3.15, permite establecer las propiedades del entorno de desarrollo integrado.

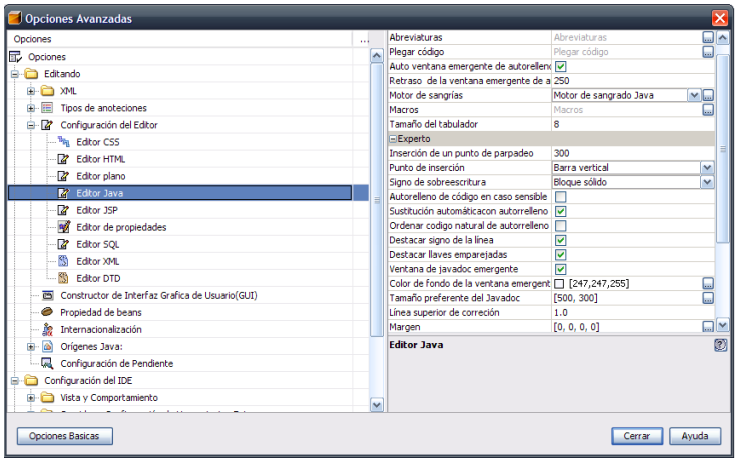


Figura 3.15 Propiedades del entorno de desarrollo

3.2.11 Otras Facilidades de NetBeans

A continuación, vamos a comentar otros elementos que se pueden tener un especial interés por la labor que desarrollan. Estos son los siguientes:

- Administrador de diseño absoluto.
- Ayuda para completar el código mientras se escribe.
- Asistente para la conexión entre componentes.
- Editor de propiedades

Ayuda para completar el código mientas se escribe.

NetBeans proporciona la característica de completar el código mientras lo escribe. Por ejemplo, según puede verse en la figura 3.16.

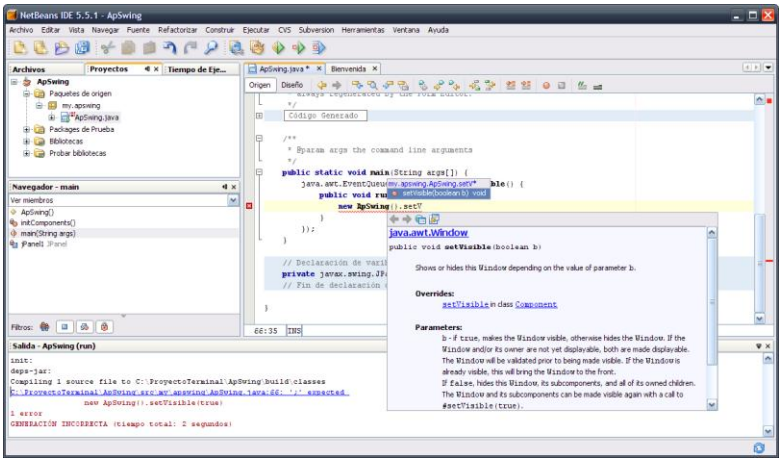


Figura 3.16 Ayuda para completar el código

Este tipo de ayuda solo estará disponible si el valor de la propiedad *Auto Popup Completion Window* tiene el valor *true*. Para verlo, ejecute la orden *Options* del menú *Tools* y seleccione *Editting>Edirtor Settings>Java Editor.*

3.3 Acceso a Base de Datos

Una base de datos es una colección de datos clasificados y estructurados que son guardados en uno o varios archivos pero referenciados como si de un único archivo se tratara. Para crear y manipular bases de datos, existen en el mercado varios sistemas administradores de bases de datos; por ejemplo, Access, SQL Server, Oracle, y DB2. Otros sistemas administradores de bases de datos de interés y de libre distribución son MySQL y PostgreSQL. MySQL es un gestor de bases de datos que cubre todas las expectativas para el desarrollo de nuestro proyecto, por lo que será el que utilizaremos.

Los datos de una base de datos relacional se almacenan en tablas lógicamente relacionadas entre sí utilizando campos clave comunes. A su vez, cada tabla dispone los datos en filas y columnas. Por ejemplo, para una lista de boletas de los alumnos de la ESIME.

Los datos relativos a la boleta (nombre, dirección, No. de boleta, etc.) son columnas que agrupamos en una fila. El conjunto de todas las filas de todos las boletas forman una tabla de la base de datos, (ver tabla 3.1).

Nombre	Dirección	No. de Boleta
Aguado Rodríguez, Jesús	Las Ramblas 3, Guadalajara	2004300854
Cuesta Suñer, Ana María	Mayor 22, México	2004300678
…	…	…

Tabla 3.1 Ejemplo de una base de datos

Una tabla es una colección de datos presentada en forma de una matriz bidimensional, donde las filas reciben también el nombre de *registros* y las columnas de *campos*.

Para crear la base, iniciamos una ventana de MySQL Query Browser

Una vez listo el administrador de bases de datos. Empezaremos por crear y seleccionar la base de datos *bd_alumno, mostrada en la figura 3.17,* esto se hace haciendo uso del siguiente comando.

```
CREATE DATABASE bd_alumnos;
```

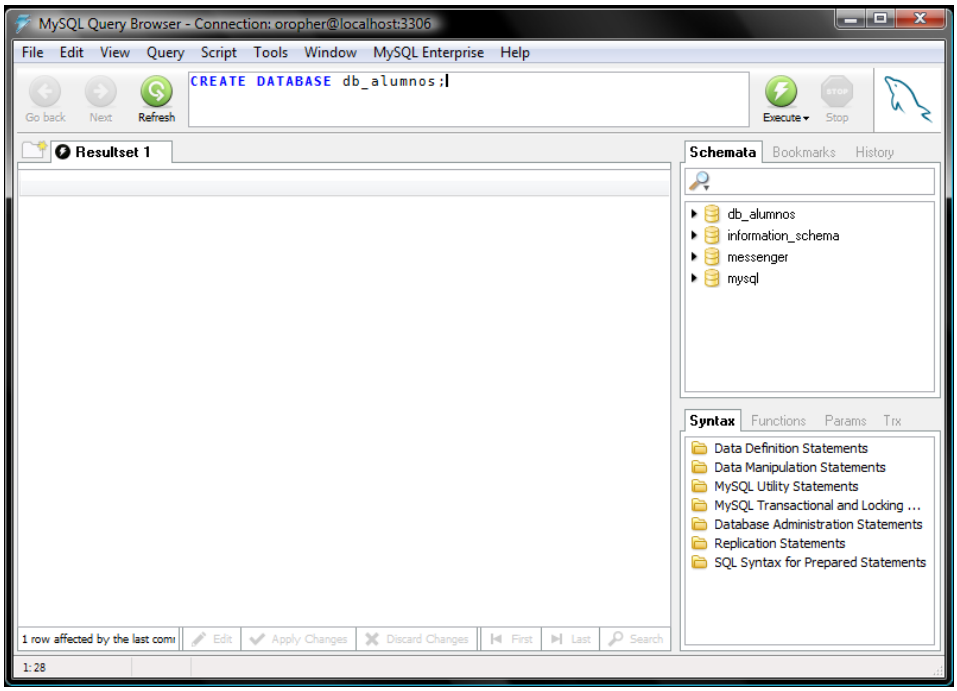


Figura 3.17 Creación de la base de datos

USE *bd_alumnos; //*ver figura 3.18

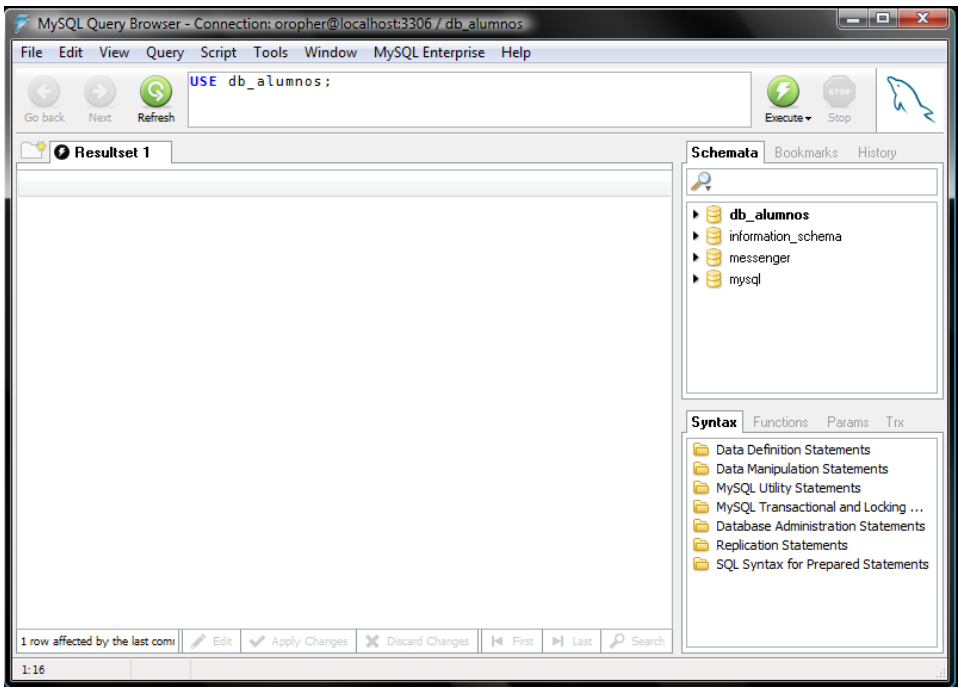


Figura 3.18 Uso de la base de datos

Puede, si lo desea mostrar las bases de datos existentes ejecutando el comando:

SHOW DATABASES; //ver figura 3.19

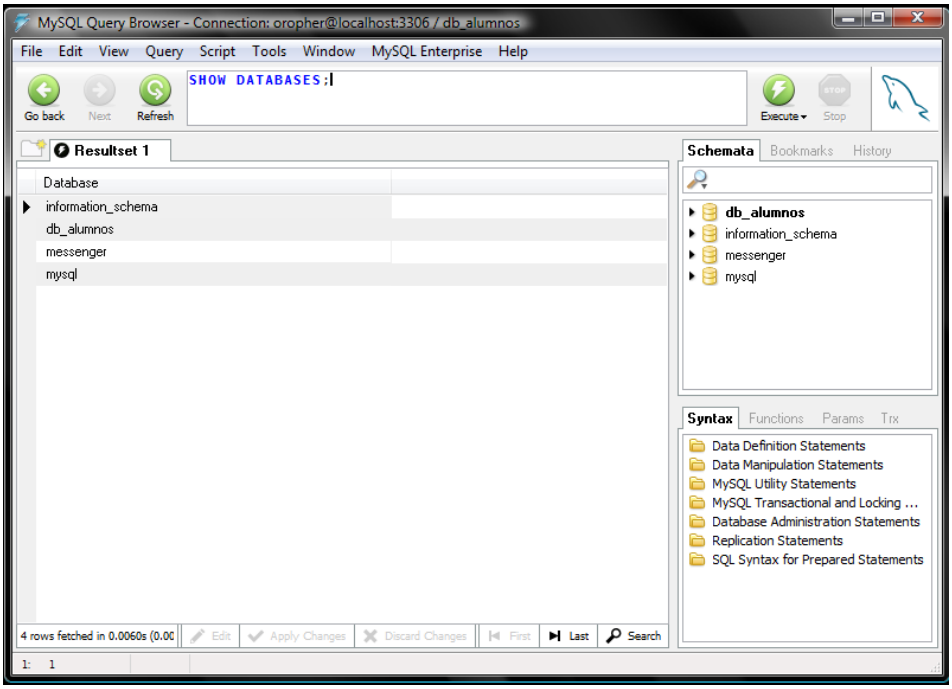


Figura 3.19 Visualización de las bases de datos existentes

Una vez creada la base de datos, añadimos la tabla alumnos. Para ello, ejecute la sentencia *CREATE TABLE*, (ver figura 3.20).

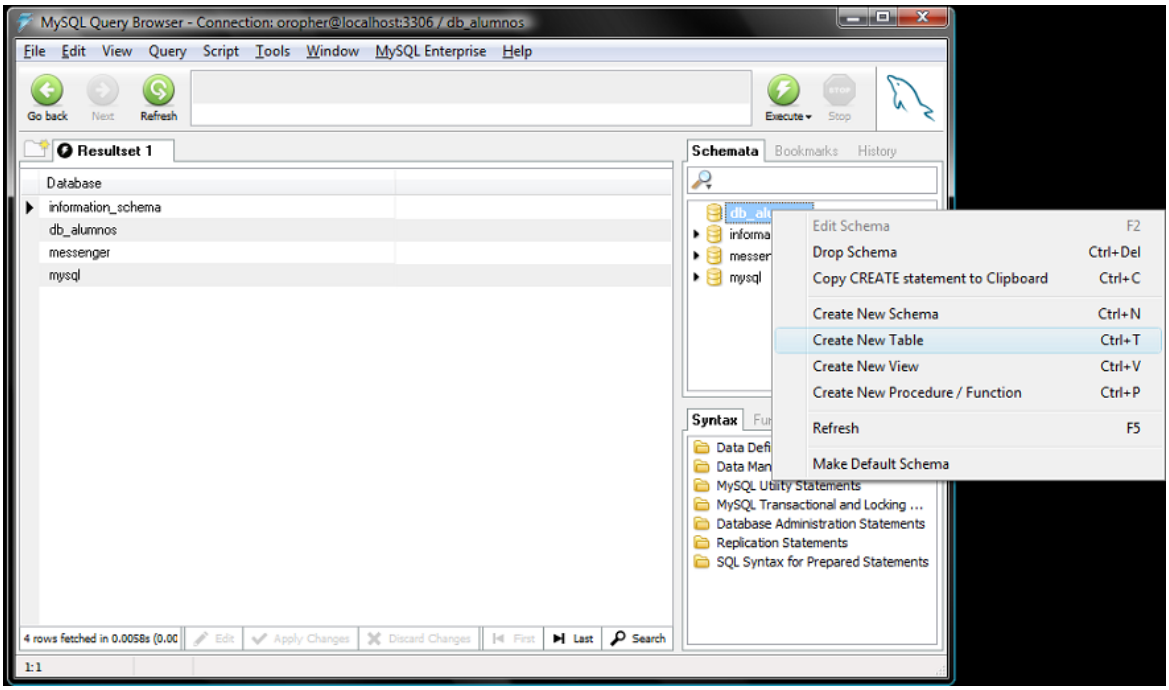


Figura 3.20 Creación de una tabla para la base de datos

Una vez seleccionada la opción `Create New Table` se abre una ventana de opciones para la creación de la tabla, esta se muestra en la figura 3.21.

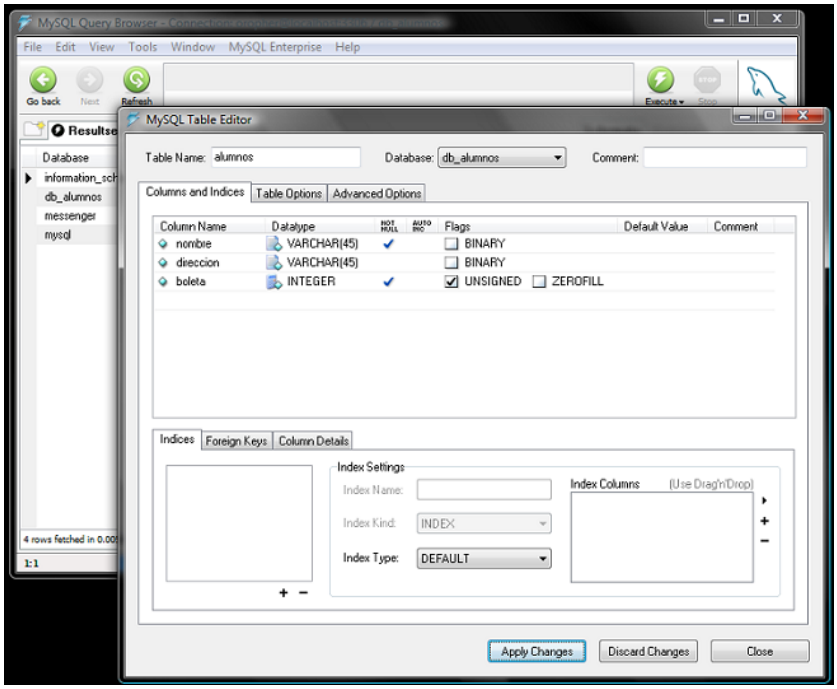


Figura 3.21 Opciones de la tabla

Para verificar la estructura de la tabla que acabamos de crear, podemos ejecutar el comando DESCRIBE, como se muestra en la figura 3.22.

```
DESCRIBE alumnos;
```

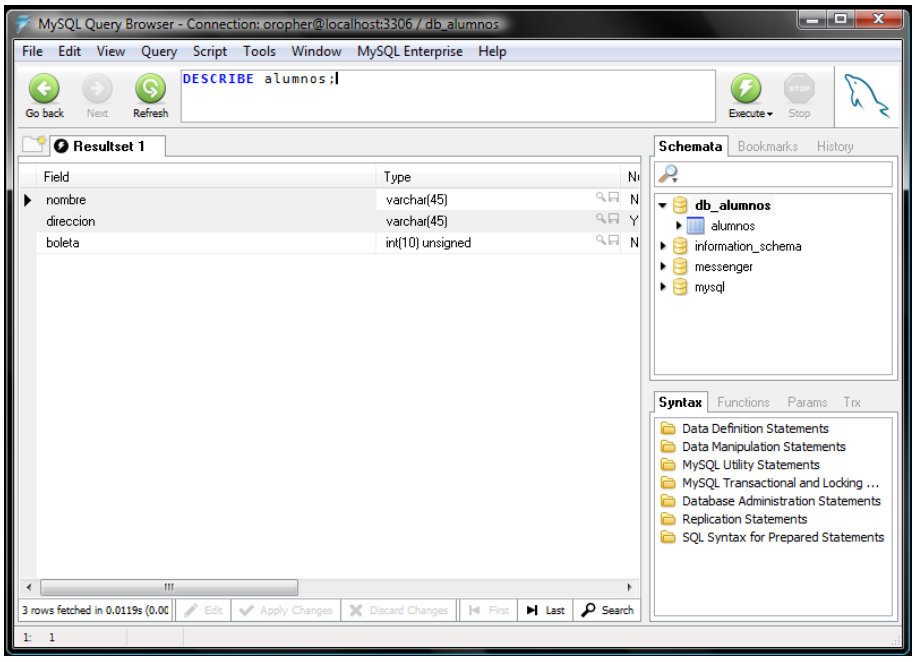


Figura 3.22 Verificación de la base de datos

También podemos mostrar las tablas de la base de datos en uso ejecutando el comando:

SHOW TABLES; //ver figura 3.23

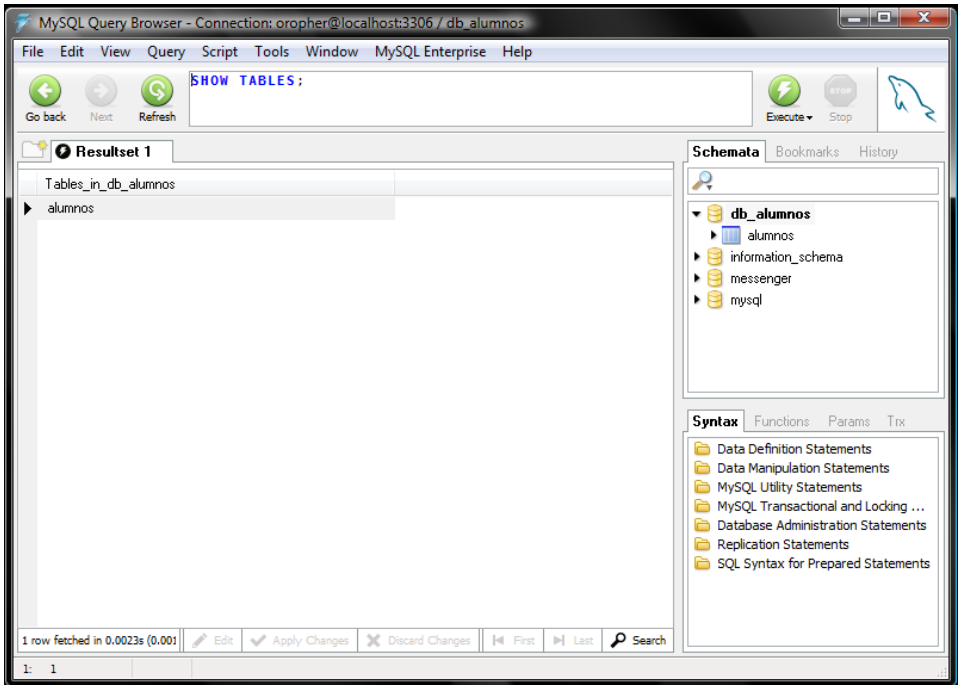


Figura 3.23 Visualizar las tablas de la base de datos

Una vez creada la tabla, se muestran los datos que se van a modificar y si están correctos los ejecuta. Esto se visualiza en la figura 3.24.

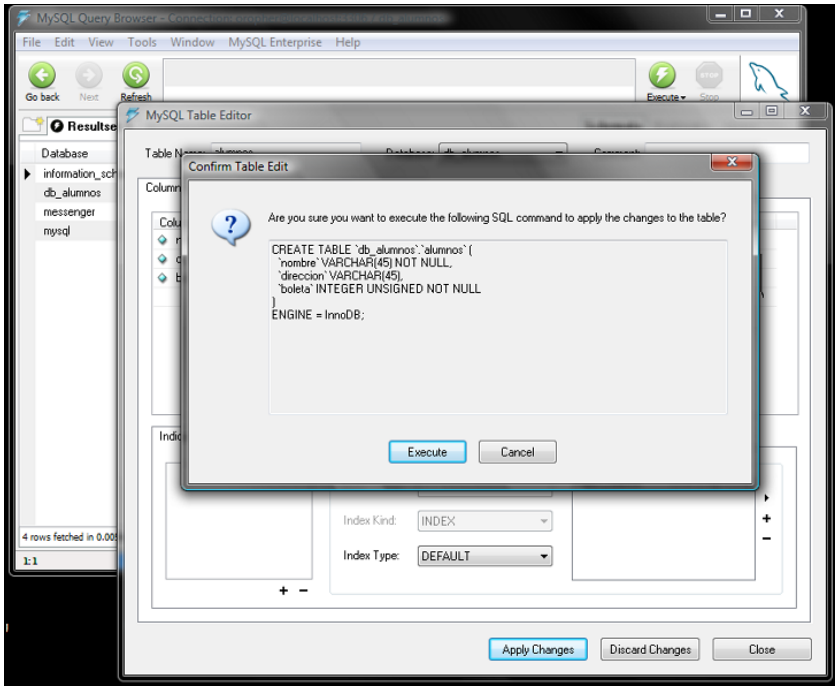
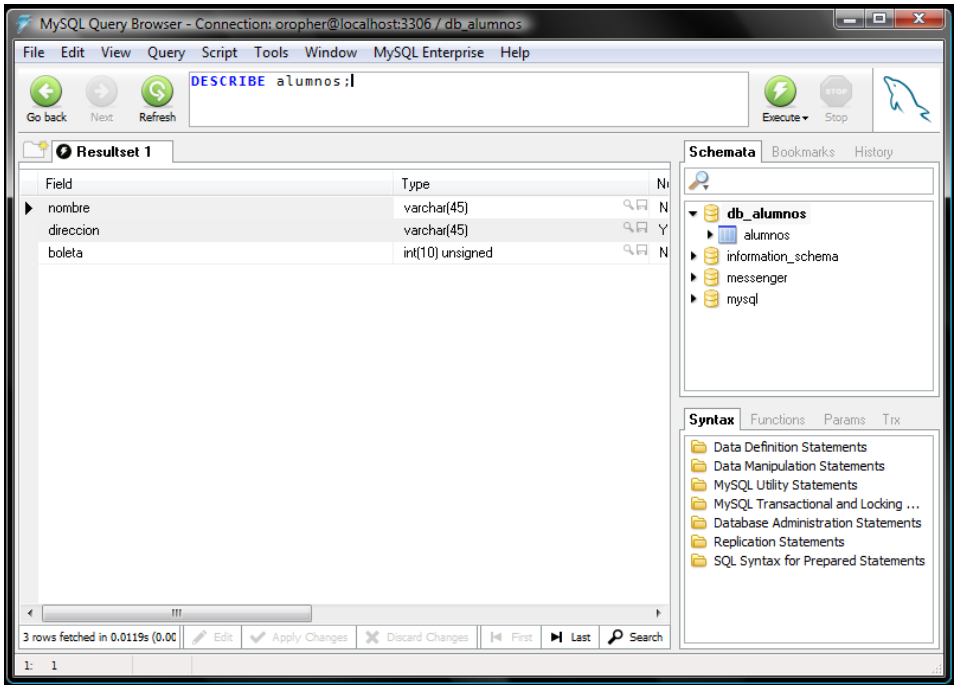


Figura 3.24 Ventana de confirmación de la creación de la tabla

La tabla resultante se visualiza en la figura 3.25.



3.25 Tabla alumnos

Para visualizar la tabla hacemos uso del comando show, esto se visualiza en la figura 3.26.

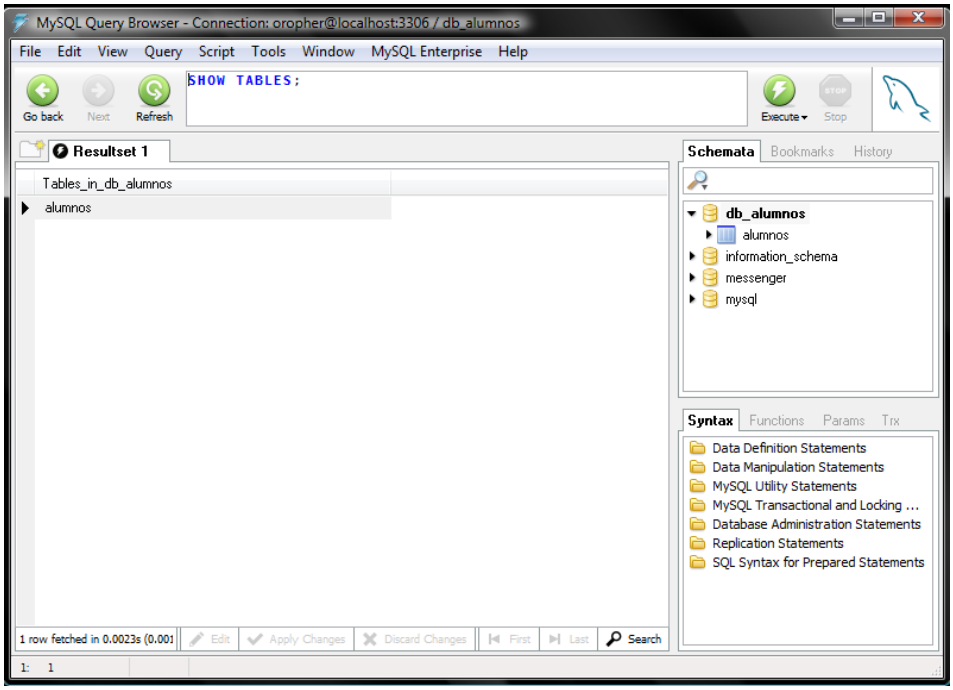


Figura 3.26 Visualizar la tabla de la base de datos usando el comando show

3.3.1 Insertar datos en la base de datos

Una vez creada la tabla alumnos, el siguiente paso es insertar filas en la misma. El código SQL que nos permite realizar esta acción es el siguiente:

```
INSERT INTO alumnos
    VALUES (’Aguado Rodríguez, Jesús’,’ Las Ramblas 3, Guadalajara’, 2004300854);
```

//ver figura 3.27

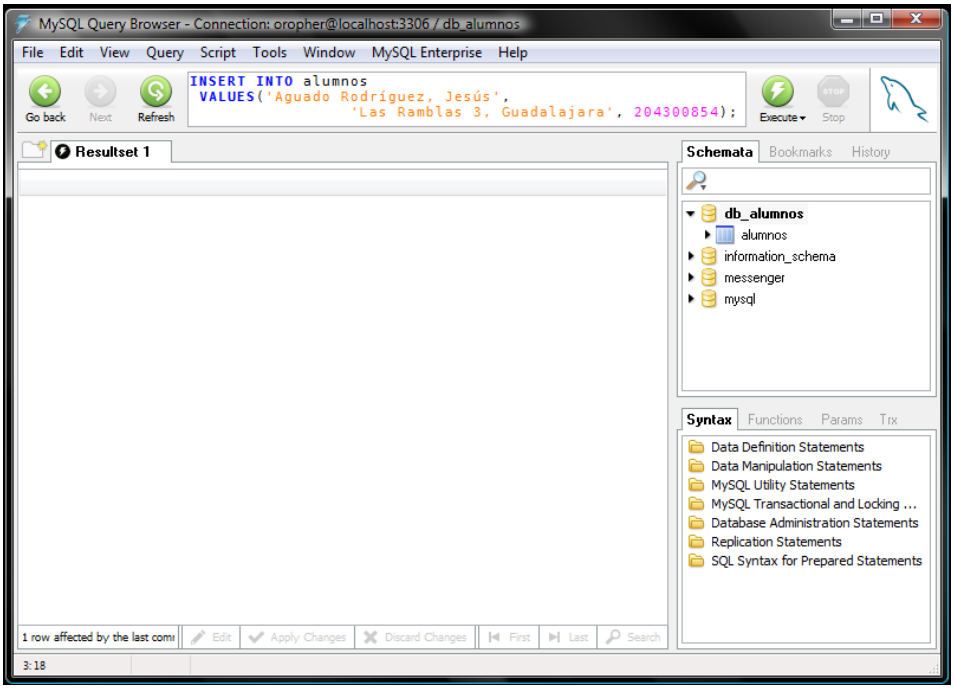


Figura 3.27 Insertar datos en la base de datos

Obsérvese que como estamos añadiendo todos los campos podemos prescindir de la lista de los nombres de las columnas. Esta orden tendríamos que ejecutarla para cada fila que quisiéramos añadir.

También es posible cargar los datos desde un archivo de texto. En este caso cada línea del archivo se debe corresponder con una fila de la tabla. Los campos de una fila deben estar separados por el carácter tabulador horizontal y a los que sean nulos (NULL) se les asignará el valor \N. Por ejemplo, suponiendo que tenemos almacenados los datos de la tabla alumnos en el archivo alumnos.txt, podemos insertarlos en dicha tabla ejecutando la siguiente orden:

```
LOAD DATA LOCAL INFILE “alumnos.txt” INTO TABLE alumnos;
```

El archivo debe estar almacenado en el directorio actual del trabajo.

3.3.2 Modificar datos en la base de datos

Supongamos que en el nombre de un determinado alumno no es correcto y tenemos que modificarlo. La forma de proceder seria como se indica a continuación:

UPDATE alumnos SET nombre = 'JOAQUIN' WHERE id_alumno=9075710; //ver figura 3.28

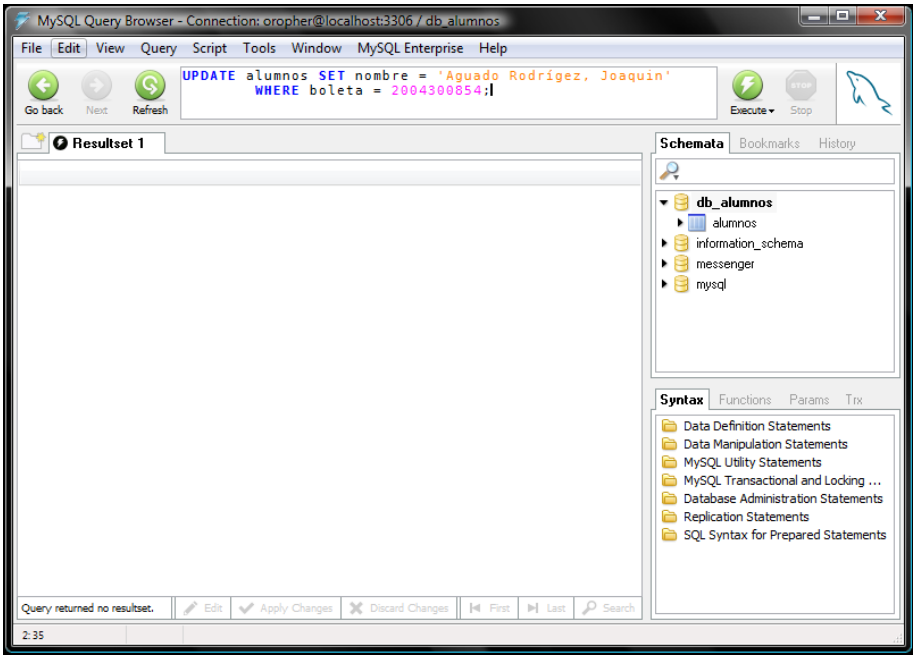


Figura 3.28 Modificar datos en la base de datos

3.3.3 Borrar registros en una tabla

Supongamos que deseamos borrar un determinado alumno. La forma de proceder sería como se indica a continuación:

DELETE FROM alumnos WHERE id_alumno=324555; //ver figura 3.29

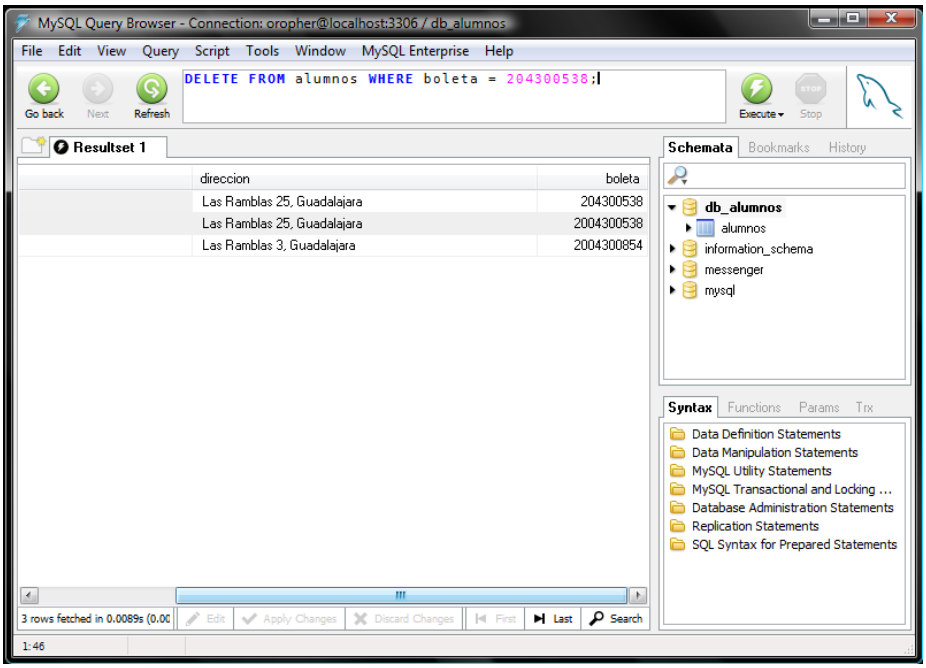


Figura 3.29 Borrar registros de una tabla

3.3.4 Obtener datos de la base de datos

Como dijimos anteriormente, supongamos que necesitamos generar un informe con los alumnos matriculados en una determinada titulación, ordenados por apellidos y que estén en un determinado curso. Este proceso se haría de la forma siguiente:

```
SELECT *FROM alumnos
    WHERE curso=1 AND titulación=12
    ORDEN BY apellidos; //ver figura 3.30
```

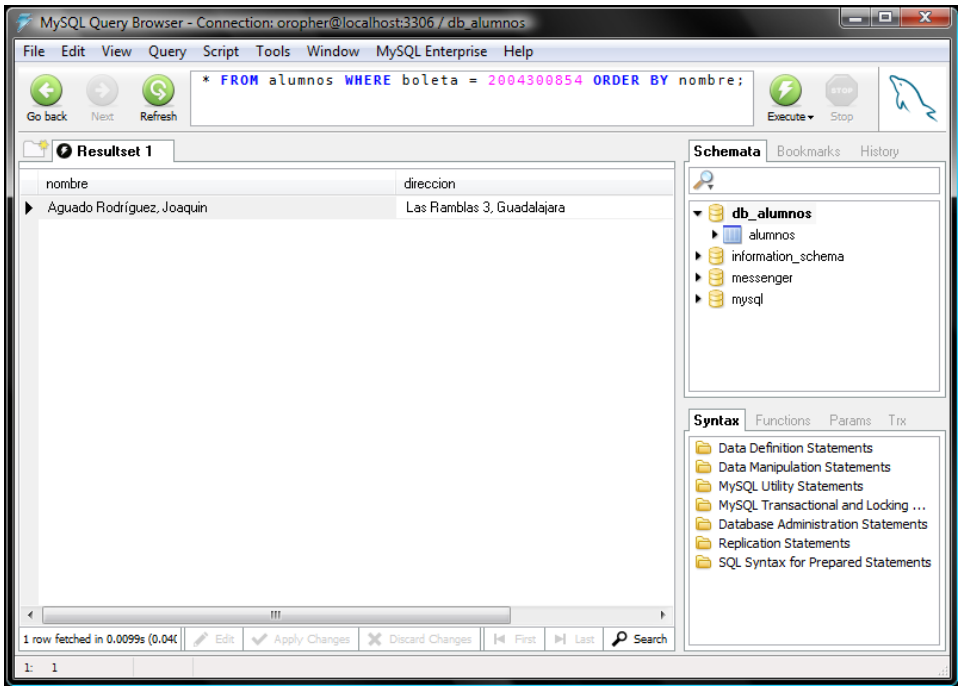


Figura 3.30 Obtener datos de la base de datos

La sentencia siguiente sería equivalente, pero especifica explícitamente que la selección será de todas las columnas de la tabla *alumnos* (alumnos.*)

```
SELECT alumnos .*FROM alumnos
    WHERE curso=1 AND titulación=12
    ORDEN BY apellidos
```

3.4 Descripción la Aplicación

El mensajero instantáneo, se compone de dos bloques principales, uno es el cliente y el segundo es el servidor.

El término cliente se refiere a aplicaciones que se ejecutan en la capa cliente (entendiendo por capa, un grupo de tecnologías que proporcionan uno o más servicios) de una aplicación de Java, en este caso nos referimos a la interfaz básica del usuario. Él es el que solicita el servicio hacia el servidor.

El servidor realiza algunas tareas en beneficio de los clientes. Algunos servicios habituales son los servicios de archivos, que permiten a los usuarios almacenar y acceder a los archivos de un ordenador y los servicios de aplicaciones, que realizan tareas en beneficio directo del usuario, (ver figura 3.31).

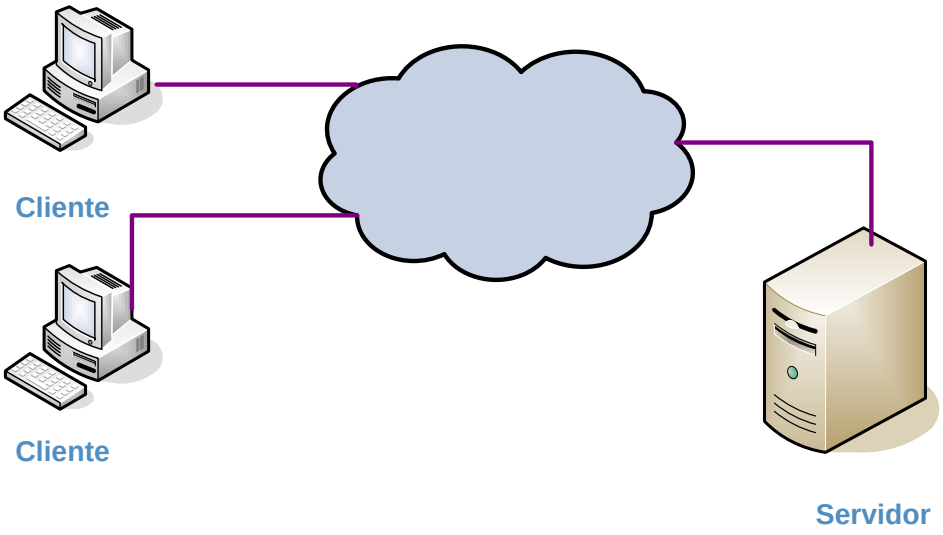


Figura 3.31 Estructura cliente-internet-servidor

De manera general en el siguiente diagrama a bloques (ver figura 3.32), se describe la forma en la que el cliente accede a la base de datos por medio del servidor, es aquí donde el cliente hace una petición al servidor el cual se encarga de realizar la petición a la base de datos y de regresar la información

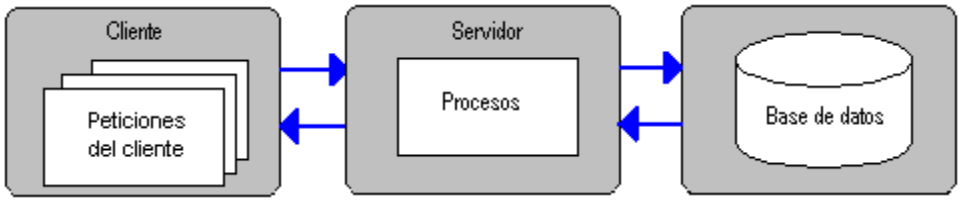


Figura 3.32 Diagrama a bloques que componen el mensajero instantáneo

Una vez que el cliente hace la petición al servidor la base de datos es la encargada de ver que el usuario al que se desea acceder efectivamente pertenezca a la base y así poder acceder a ella y poder establecer comunicación los demás usuarios

En la figura 3.33 se muestra el diagrama a bloques por parte del servidor y por parte del cliente en la figura 3.34, los cuales nos darán una idea de cómo se realizará el proyecto de manera general.

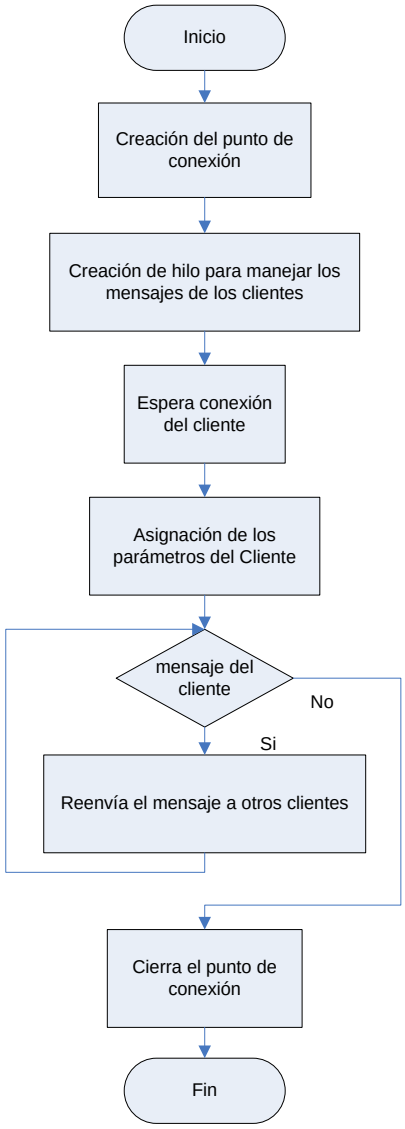


Figura 3.33 Diagrama de flujo del servidor

Creación del punto de conexión. Consiste en abrir un puerto por el que se va a establecer la comunicación con el exterior y preparar la aplicación y preparar la aplicación para recibir la información que los clientes le envíen.

Creación de hilo para manejar los mensajes de los clientes. Se prepara la aplicación para recibir a los nuevos clientes entrantes y crea un hilo para cada uno de ellos por medio de los cuales se pueden manejar a cada uno de ellos y no uno por uno.

Espera conexión del cliente. Se mantiene en espera de conexiones entrantes (clientes).

Asignación de los parámetros del Cliente. Una vez que un cliente nuevo se conecta, se leen las peticiones para asignar sus parámetros correspondientes como por ejemplo su nombre de usuario.

Mensaje del cliente. Una vez establecida la conexión se mantiene en espera de que un cliente mande un mensaje; si es así, es reenviada a su destinatario correspondiente. Si no es enviado ningún mensaje en un determinado tiempo se cierra el punto de conexión y termina la aplicación.

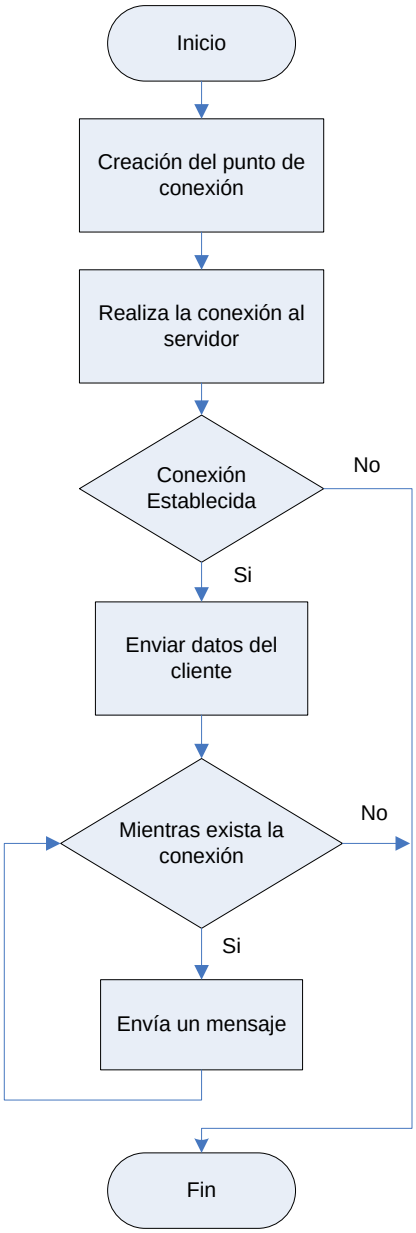


Figura 3.34 Diagrama de flujo del cliente

Creación del punto de conexión. Se prepara para comunicarse con el servidor.

Realiza la conexión al servidor. Se comunica hacia la dirección del servidor por un puerto determinado

Conexión establecida. El cliente envía los datos del usuario para ser establecidos en la parte del servidor.

Enviar datos del cliente. Mientras exista la conexión; se realiza el envió de información hacia el servidor para que la pueda enviar al usuario correspondiente, hasta que se dejen de enviar mensajes, si es así, se termina la conexión y se cierra el punto de conexión.

3.5 Funcionamiento de la Aplicación

Como primer punto, partimos de la ventana principal que se visualiza al ejecutar el mensajero, la cual muestra en pantalla los parámetros de usuario y contraseña, como se muestra en la figura 3.35.

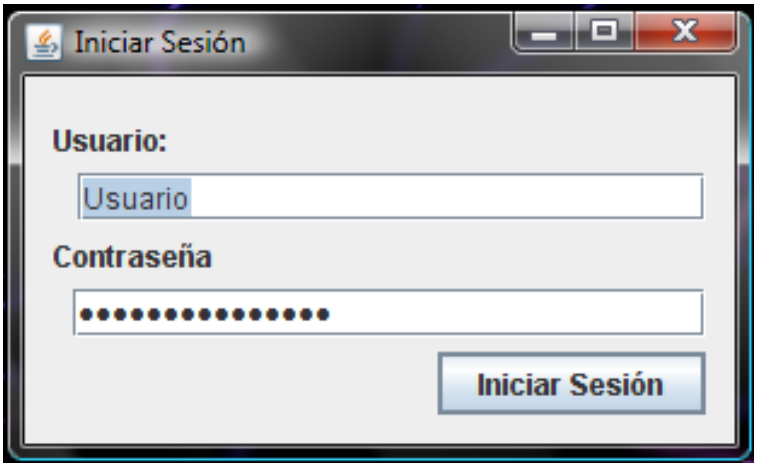


Figura 3.35 Ventana de inicio del Mensajero

En esta ventana, el usuario se introduce de manera independiente a su sesión accediendo con su cuenta propia.

Código de inicio de sesión

```
/*Se le envia el nombre de usuario y sontraseña ingresadas por el alumno y
son comparadas con el contenido de la base de datos para poder dar acceso a
la cuenta del mismo.*/

if(DB.IniciarSesion(user.getText(),pass.getText()))
{
    this.setVisible(false);
    Contactos X = new Contactos(usuario,password);
    A.setVisible(true);
}
else
{
    error.setText("Usuario o contraseña invalidos!!!");
}
```

Una vez iniciada la sesión, se muestra la ventana de bienvenida (ver figura 3.36) en donde Nick que se le asigna es el mismo que el nombre del usuario, en pocas palabras es el nombre con el cual el usuario se da a conocer.

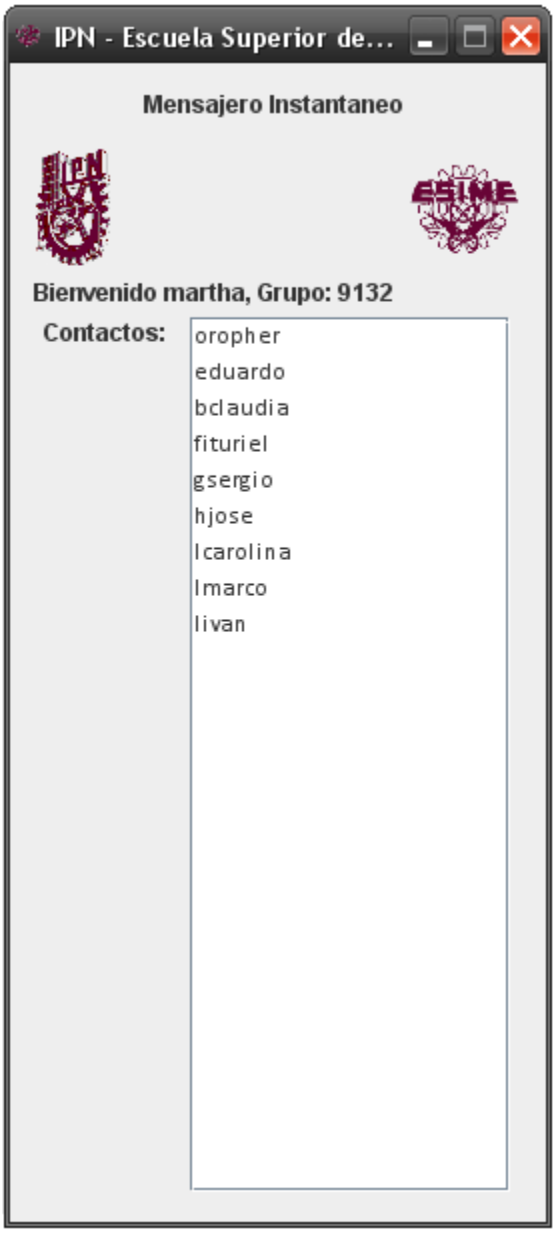


Figura 3.36 Ventana de petición de identificador de usuario.

Código de petición de los contactos del usuario

```
/* Una vez que se acceso al mensajero se busca dentro de la base de datos
los contactos del usuario que en este caso serán los miembros de su grupo y
los profesores asignados en el semestre en el que se encuentre. */

public void Iniciar(String user)
    {
        lista = db.Contactos(ID=db.getID(user));
        Iterator it = lista.iterator();
        while (it.hasNext())
        {
            Contact.addElement((String)it.next());
        }
// Se la da bienvenida al usuario con la información obtenida de la base de
// datos.
        grupo = db.getGrupo(user,ID);
        bienvenida.setText("Bienvenido "+usuario+",\n Grupo: "+grupo);
        Contactos.setModel(Contact);
        this.setVisible(true);
    }
```

En la figura 3.37 se muestra la ventana con la cual se inicia la conversación:

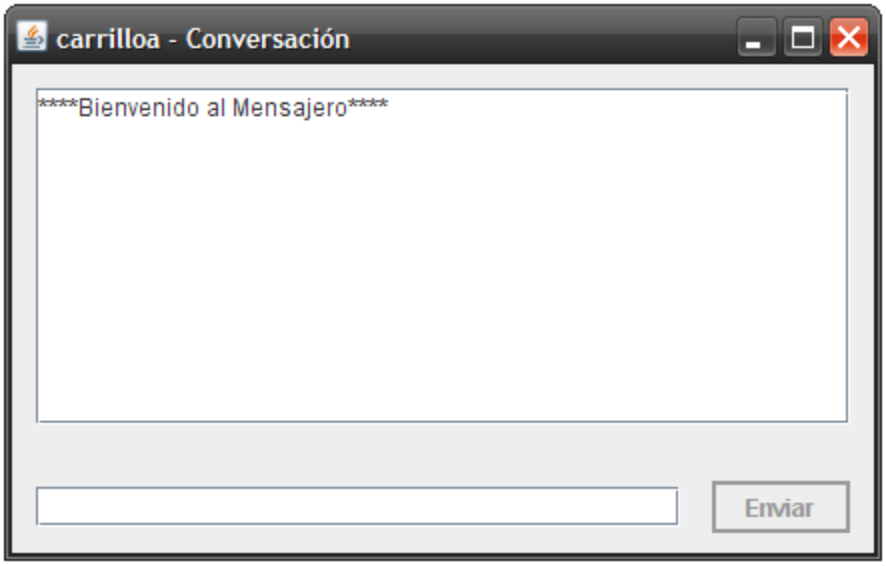


Figura 3.37 Ventana de bienvenida al mensajero

Código de asignación del Nickname

```
/* Se hace la asignación del nombre de usuario y al final se le agrega el numero de
identificación con el cual se garantiza que el mensaje solo llegue a este usuario */

 if(ID<10)
        {
            salida.println(user1+"0"+db.getID(user2.substring(0,user2.length()-2)));
        }
        else
        {
            salida.println(user1+db.getID(user2.substring(0,user2.length()-2)));
        }
```

Una vez que los demás usuarios han iniciado sesión, es posible establecer comunicación entre ellos, la ventana mostrada en la figura 3.38 representa a dos usuarios conversando.

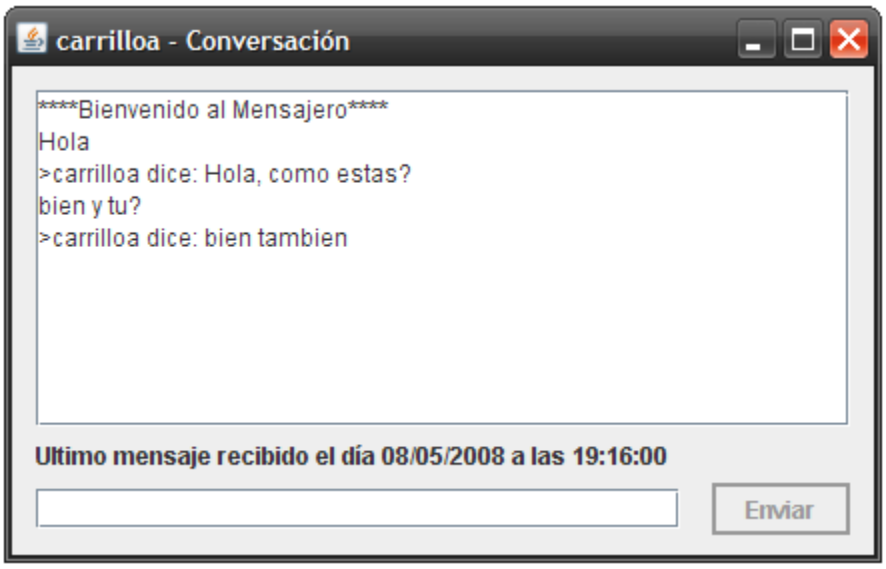


Figura 3.38 Ventana de conversación

Código del envió de mensajes

```
/* Se realize el  envio el mensaje, pasándole como parámetro el usuario al que va dirigido para
que este mensaje sea solamente enviado a él  */

salida.println("#"+user2+" "+Texto.getText());
Mensajes.setText(Mensajes.getText()+Texto.getText()+System.getProperty("line.separator"));
Texto.setText("");
```

En la figura 3.39, se muestra la fecha y la hora en la que está llegando el último mensaje que ha sido enviado

Figura 3.39 Hora y Fecha del último mensaje recibido

Código para la Fecha y Hora del último mensaje recibido

```
/* Se crea un objeto tipo Calendar y se obtienen los datos necesarios de
este. */

 calendario = new GregorianCalendar();
 anho = ""+calendario.get(Calendar.YEAR);
 if(calendario.get(Calendar.MONTH)<10)
       mes = "0"+calendario.get(Calendar.MONTH);
else
       mes = ""+calendario.get(Calendar.MONTH);
if(calendario.get(Calendar.DAY_OF_MONTH)<10)
       dia = "0"+calendario.get(Calendar.DAY_OF_MONTH);
else
       dia = ""+calendario.get(Calendar.DAY_OF_MONTH);
if(calendario.get(Calendar.HOUR_OF_DAY)<10)
       hora = "0"+calendario.get(Calendar.HOUR_OF_DAY);
else
       hora = ""+calendario.get(Calendar.HOUR_OF_DAY);
if(calendario.get(Calendar.MINUTE)<10)
       minutos = "0"+calendario.get(Calendar.MINUTE);
else
       minutos = ""+calendario.get(Calendar.MINUTE);
if(calendario.get(Calendar.SECOND)<10)
       segundos = "0"+calendario.get(Calendar.SECOND);
else
       segundos = ""+calendario.get(Calendar.SECOND);
clienteChat.setUltimoMsg("Ultimo mensaje recibido el día "+ dia + "/" + mes
+ "/" + anho +" a las " + hora + ":" + minutos + ":" + segundos );
```

En la figura 3.40 se muestra una conversación con más de dos usuarios en la cual el mensaje que se desea enviar a un usuario en específico le llega solo a él, esto se hace seleccionando el usuario al que se le desea enviar el mensaje y si no es así, el mensaje enviado le llega a todos los usuarios conectados.

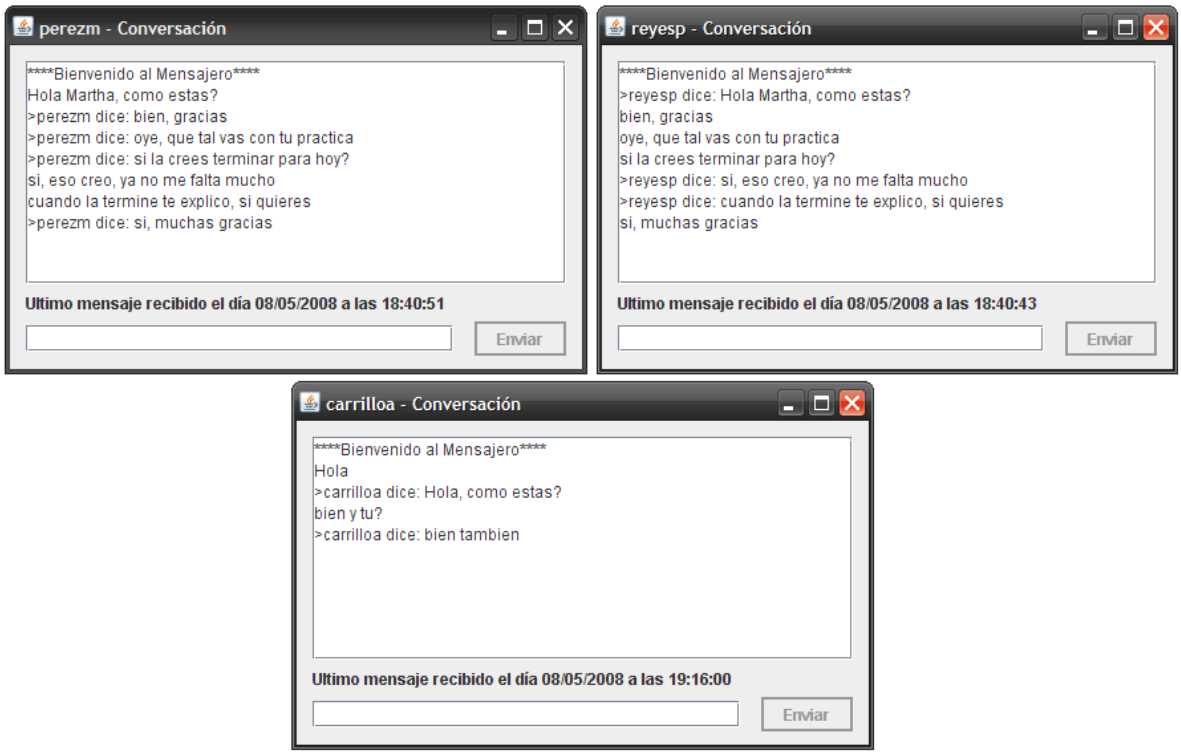


Figura 3.40 Ventana de mensajes específicos.

Con el comando `/listar`, es posible enlistar a los usuarios que se encuentran conectados. La figura 3.41 nos muestra la forma en la que lo podemos hacer.

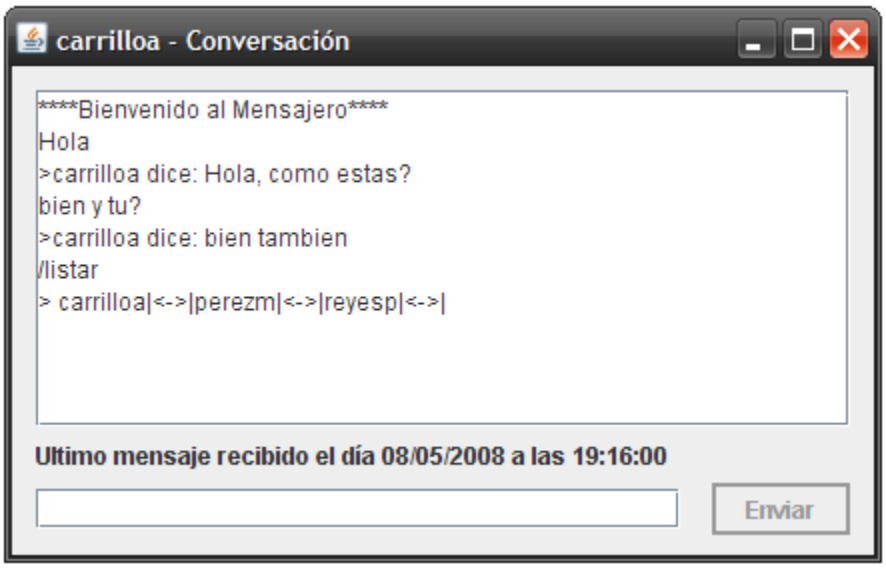


Figura 3.41 Ventana de la lista de los usuarios conectados

Comandos para listar y desconectar

```
/* Se define el commando desconectar que al ser ingresado termina la
comunicación de esta ventana. */

 if((mensaje.equals("/desconectar"))){
       salida.println("Hasta luego,,,");
        break;

/* Se define el commando listar que al ser ingresado lista los usuarios que
se encuentran conversando en una ventana. */

 } else if((mensaje.equals("/listar"))){
       escribeUsuario(this,"> "+listaUsuarios());
 }
```

Para abandonar la conversación, dicho de otra forma, salir de la sesión, se hace uso del comando `/desconectar`, el cual después de unos segundos, se encarga de cerrar la ventana de conversación, esto se muestra en la figura 3.42.

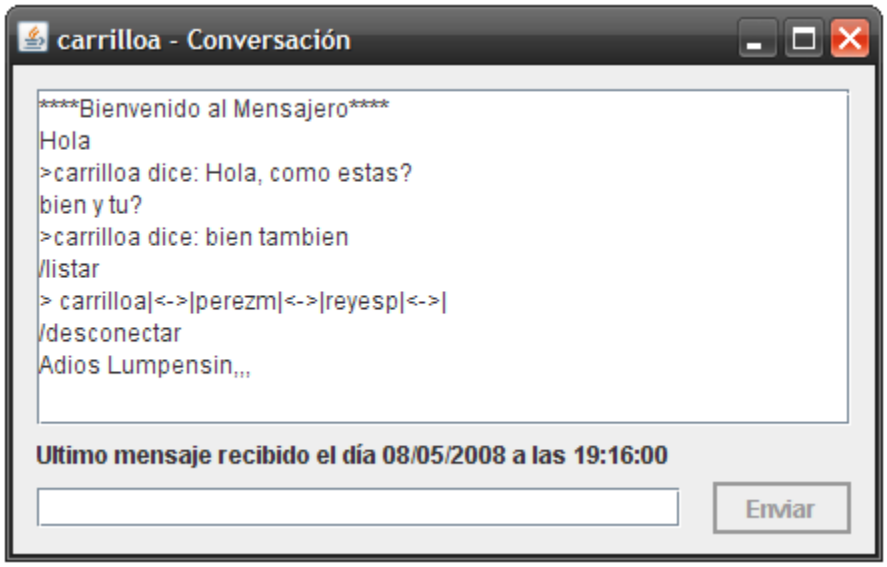


Figura 3.42 Ventana de término de la sesión

A continuación, se agrego la parte de sonido a los mensajes, para avisar a cada usuario cuando está siendo solicitado o tiene un nuevo mensaje.

Código para reproducir sonido dentro de la aplicación.

```
/*Se hizo uso de la librería JLayer para reproducir sonido dentro de la
aplicacion. */

Player player;
     try {
        FileInputStream fis = new FileInputStream("C:" + File.separatorChar +
File.separatorChar + "mess.mp3");
        player = new Player(fis);
        player.play();
     } catch (JavaLayerException ex) {
        Logger.getLogger(Sesion.class.getName()).log(Level.SEVERE, null, ex);
     } catch (IOException ioe) {
        ioe.printStackTrace();
     }
```

Cuando excedemos el tamaño del cuadro de texto destinado a la conversación, aparece una barra de desplazamiento horizontal, para poder desplazarnos a través de todo el contenido del cuadro del texto. (Ver figura 3.43)

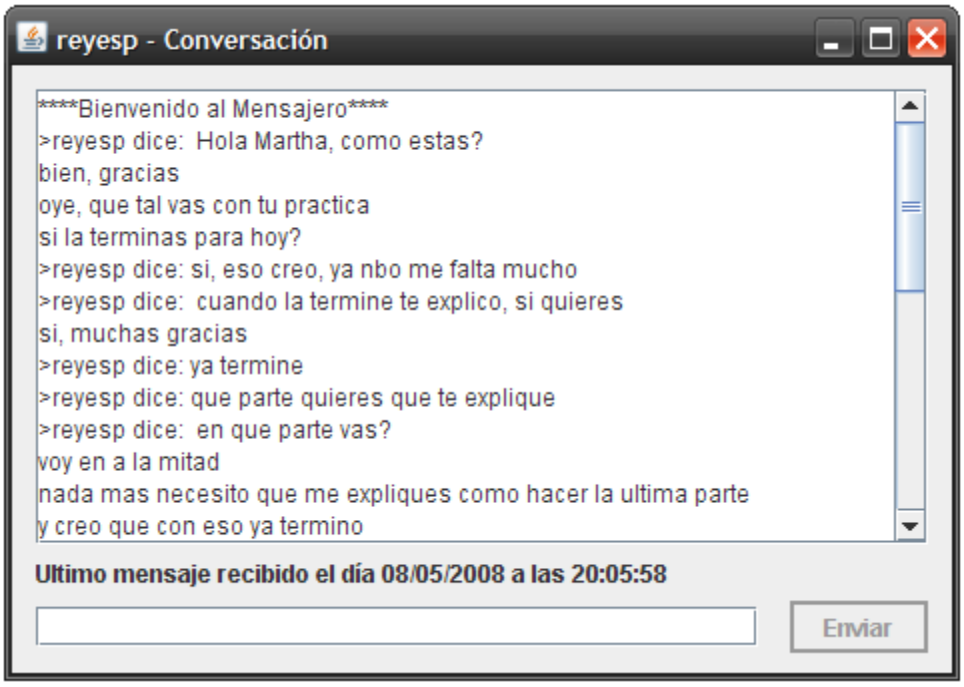


Figura 3.43 Ventana que muestra la barra de desplazamiento vertical.

En el caso del servidor, hacemos uso de una ventana de comandos (ver figura 3.42), el cual es de gran importancia, ya que es el que se encarga de dar acceso y respuesta a las peticiones de los usuarios. La siguiente ventana nos muestra, la forma en la que trabaja el servidor una vez que los dos usuarios establecieron su sesión.

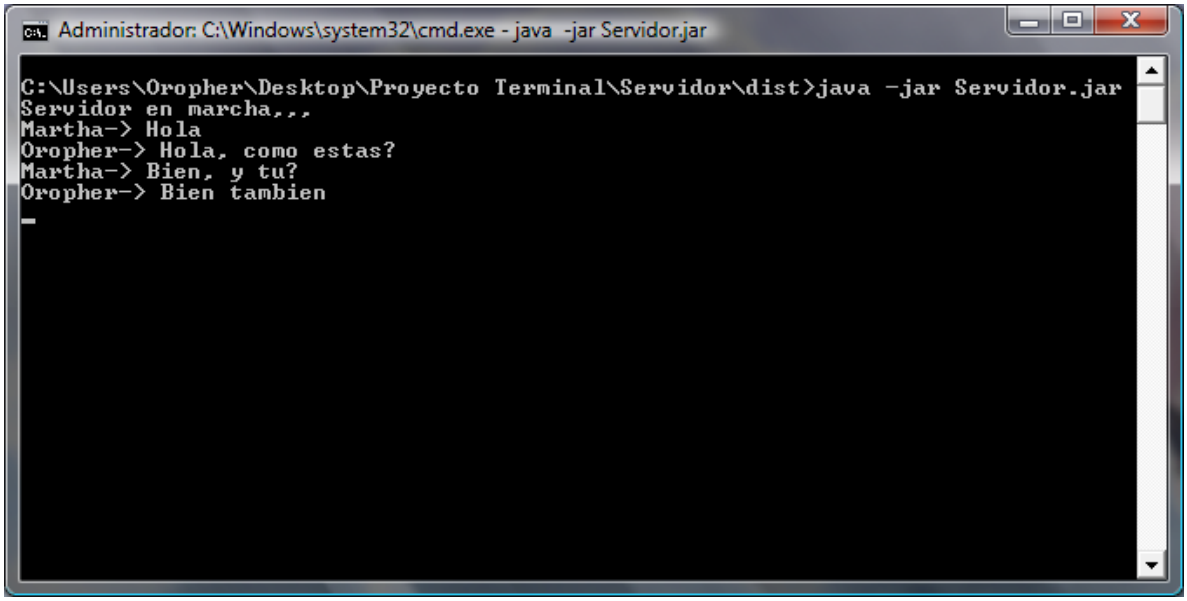


Figura 3.42 Ventana del Servidor

Código del servidor

```
/* Se comprueba que el texto recibido desde el cliente es un mensaje
dirigido a un usuario en específico y el mensaje es redirigido al
destinatario. */

else if(mensaje.startsWith("#")){
                    String palabras[]=mensaje.split(" ");
                    Iterator itera = Clientes.iterator();
                    while(itera.hasNext()){
                        HiloCliente temp = (HiloCliente)itera.next();
                        String nick =
palabras[0].substring(1,palabras[0].length());
                        if(temp.Nick.equalsIgnoreCase(nick)) {
                            String msj =
this.Nick.substring(0,this.Nick.length()-2) + " dice: " +
mensaje.substring(palabras[0].length(),mensaje.length());
                            escribeUsuario(temp,">>> "+msj);
                        }
                    }
                }
```

Capítulo IV

Costos

El objetivo de este capítulo es hacer un análisis, acerca de los costos que presenta nuestro proyecto y así poder saber si es viable o no, de tal forma que nos da un panorama de las condiciones en las que realizaría el trabajo.

4.1 Cálculos del costo del diseño del proyecto

Para tener un estimado de los costos se toma en consideración el salario de 56.286 dolares/año (que equivalen a 591,003 pesos), que es el salario promedio de un programador y luego se multiplica ese resultado por 2,40 que incluye cualquier gasto extra diferente de los programadores como pueden ser: luz, teléfono, papelería, etc.

El análisis de acuerdo al modelo de estimación de costos COCOMO con el que se calcula un estimación del costo de producción del software bajo un modelo de Ingeniería de Software cerrado. Aplicando la herramienta SLOCCount:

Concepto	**Valor**
Total de líneas de código	90,812
Años de desarrollo	1.76
Esfuerzo persona-año	12.96
Costo estimado de desarrollo	3,073,916 dólares 32,267,118 pesos

Figura 4.1 Estimación de costos en Javac

Lenguaje	**Líneas de código**	**%**
Java	88,346	97,28%
Sh	2,466	0,10%

Figura 4.2 Líneas de código en lenguajes de programación Javac

4.2 COCOMO II

COCOMO II es un modelo que permite estimar el costo, esfuerzo y tiempo cuando se planifica una nueva actividad de desarrollo software. Está asociado a los ciclos de vida modernos. El modelo original COCOMO ha tenido mucho éxito pero no puede emplearse con las prácticas de desarrollo software más recientes tan bien como con las prácticas tradicionales. COCOMO II apunta hacia los proyectos software de los 90's y de la primera década del 2000, y continuará evolucionando durante los próximos años.

En resumen, los objetivos a la hora de la creación del modelo COCOMO II fueron:

- Desarrollar un modelo de estimación de tiempo y de costo del software de acuerdo con los ciclos de vida utilizados en los 90´s y en la primera década del 2000.
- Desarrollar bases de datos con costos de software y herramientas de soporte para la mejora continua del modelo.
- Proporcionar un marco analítico cuantitativo y un conjunto de herramientas y técnicas para la evaluación de los efectos de la mejora tecnológica del software en costos y tiempo del ciclo de vida software.

Estos objetivos apoyan las necesidades primarias expresadas por los usuarios de la estimación de costes del software. En orden de prioridades, estas necesidades eran: el apoyo de la planificación de proyectos, la previsión de personal del proyecto, la preparación del proyecto, la replanificación, el seguimiento del proyecto, la negociación del contrato, la evaluación de la propuesta, la nivelación de recursos, exploración de conceptos, la evaluación del diseño referentes a la oferta/demanda. Para cada una de estas necesidades COCOMO II proporcionará un apoyo más moderno que sus predecesores, el COCOMO original y Ada COCOMO.

Para apoyar a los distintos sectores del mercado software, COCOMO II proporciona una familia de modelos de estimación de coste software cada vez más detallado y tiene en cuenta las necesidades de cada sector y el tipo de información disponible para sostener la estimación del costo del software. Esta familia de modelos está compuesta por tres submodelos cada uno de los cuales ofrece mayor fidelidad a medida que uno avanza en la planificación del proyecto y en el proceso de diseño. Estos tres submodelos se denominan:

- El modelo de Composición de Aplicaciones. Indicado para proyectos construidos con herramientas modernas de construcción de interfaces gráficos para usuario.
- El modelo de Diseño anticipado. Este modelo puede utilizarse para obtener estimaciones aproximadas del costo de un proyecto antes de que esté determinada por completo su arquitectura. Utiliza un pequeño conjunto de drivers de coste nuevo y nuevas ecuaciones de estimación. Está basado en Punto de Función sin ajustar o KSLOC (Miles de Líneas de Código Fuente).

Basándonos en este modelo, el cálculo de los costos para nuestra aplicación queda de la siguiente forma:

Concepto	**Valor**
Líneas de Código del Servidor	178
Líneas de Código de la Base de Datos	60
Líneas de Código de la pantalla de inicio de sesión	159
Líneas de Código de la ventana de conversación	183
Líneas de Código del función principal	28
Total de líneas de código	608
Años de desarrollo	1
Esfuerzo persona-año	7.36
Costo estimado de desarrollo	20,580 dólares 216,093 pesos

Figura 4.3 Estimación de los costos totales del proyecto

De acuerdo con el Organismo de Certificación OCC, el salario que gana un programador de Java (Ver Anexo [2]) es de 132, 000 pesos correspondientes a un año, que es el tiempo de desarrollo de la aplicación sin considerar los costos de las licencias de los programas utilizados.

Capítulo V

Resultados

En este capítulo se presentan los resultados que en principio se obtuvieron, las respuestas al implementarlo y su utilidad al ejecutarlo.

Las pruebas realizadas han sido en el sistema operativo Windows XP y Windows Vista, en los cuales se han obtenido los siguientes resultados:

- Se agrego soporte en la base de datos, para crear las cuentas de los usuarios, con ello es posible crear cuentas para cada uno de los alumnos con sus respectivos profesores, (ver figura 5.1).

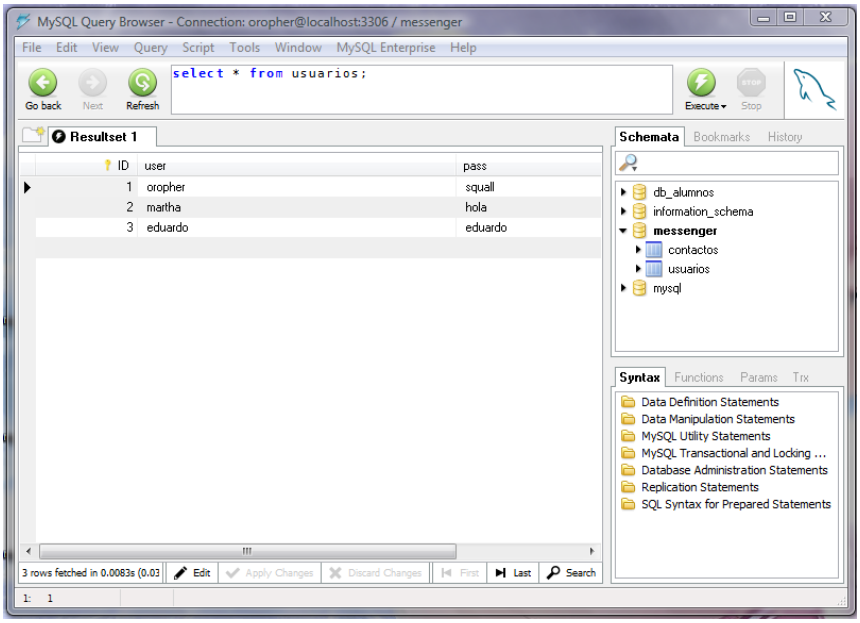


Figura 5.1 Cuentas asignadas a los usuarios

- La comunicación entre usuarios responde de manera adecuada dentro de las conversaciones y de manera segura, ya que cada uno de los usuarios cuenta con una cuenta exclusiva de acceso al mensajero, (ver figura 5.2).

Figura 5.2 Ventana de inicio de sesión

- El servidor presenta una buena respuesta al hacer uso del mensajero desde varios puntos de la red, (ver figura 5.3).

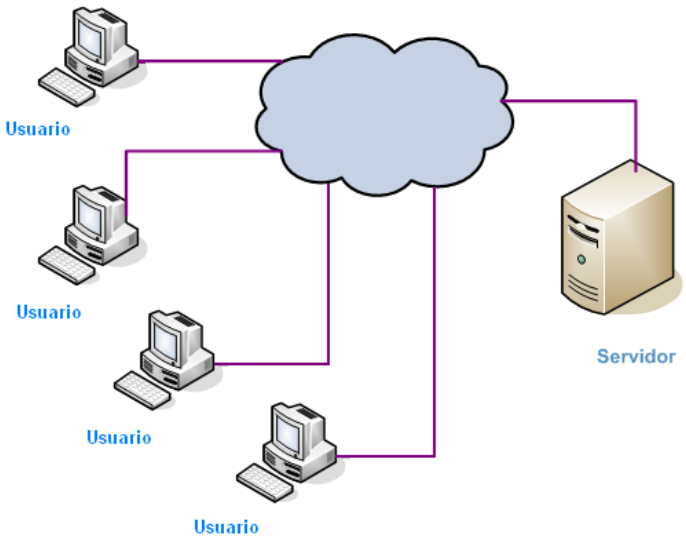


Figura 5.3 Respuesta del servidor al acceder desde diferentes puntos de la red

- Las pruebas hechas al ejecutar el mensajero se hicieron primero de manera local y una vez que este respondió como se esperaba, se ejecuto vía internet y como respuesta fue posible establecer comunicación entre los usuarios.

Las pruebas realizadas en el sistema operativo Linux dentro de nuestra aplicación, se han obtenido los siguientes resultados:

- Se realizo la instalación de la maquina virtual de java en diferentes Sistemas Operativos Linux para verificar su correcto funcionamiento

Al implementar la tecnología de java.nio, se obtuvieron los siguientes resultados:

- Gracias a esta tecnología se ha agregado soporte para cientos o miles de usuarios dependiendo del servidor que se utilice, en contraste con los sockets que solo soportan algunas decenas de usuarios.

Observaciones al desarrollar la aplicación.

Para cada uno de los resultados obtenidos anteriormente se tuvieron que tomar ciertas consideraciones y alternativas para poder dar un buen funcionamiento al ejecutar el mensajero.

- Para el primer punto, se presentaron algunos problemas de conexión al momento de dar acceso a los usuarios y esto se solucionó dando permisos especiales de acceso a la cuenta encargada de manejar la base de datos dentro del servidor.

- Para el tercer punto, el servidor se tuvo que configurar fuera de la red de donde se encontraba, en una zona desmilitarizada para que tuviera salida directa a internet y así poder dar servicio a los usuarios.

- Los problemas presentados para el punto cuatro fue que como ya se mencionó anteriormente, se tuvo que dar permisos de acceso.

Una forma más visual de los resultados obtenidos hasta el momento, considerando todos los problemas que surgieron durante su desarrollo se muestra en la figura 5.4.

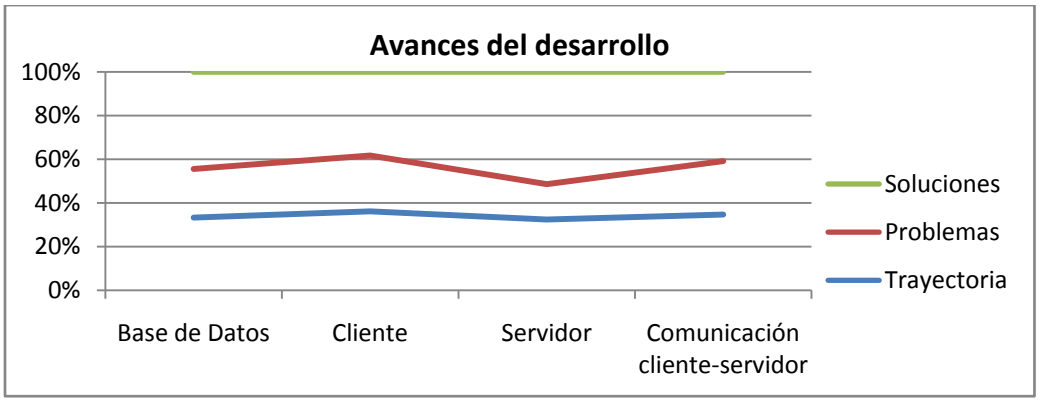


Figura 5.4 Respuesta a los avances del desarrollo del proyecto

Dentro de las pruebas realizadas, considerando que se hicieron en 20 equipos de cómputo via internet se obtuvieron los siguientes resultados que difieren de la versión de la aplicación generada.

Para la versión 1 en principio, como ya se mencionó anteriormente, los problemas de acceso a la base de datos estaban restringidos y por lo tanto no hubo acceso a la red y así no existieron resultados satisfactorios, (ver figura 5.5).

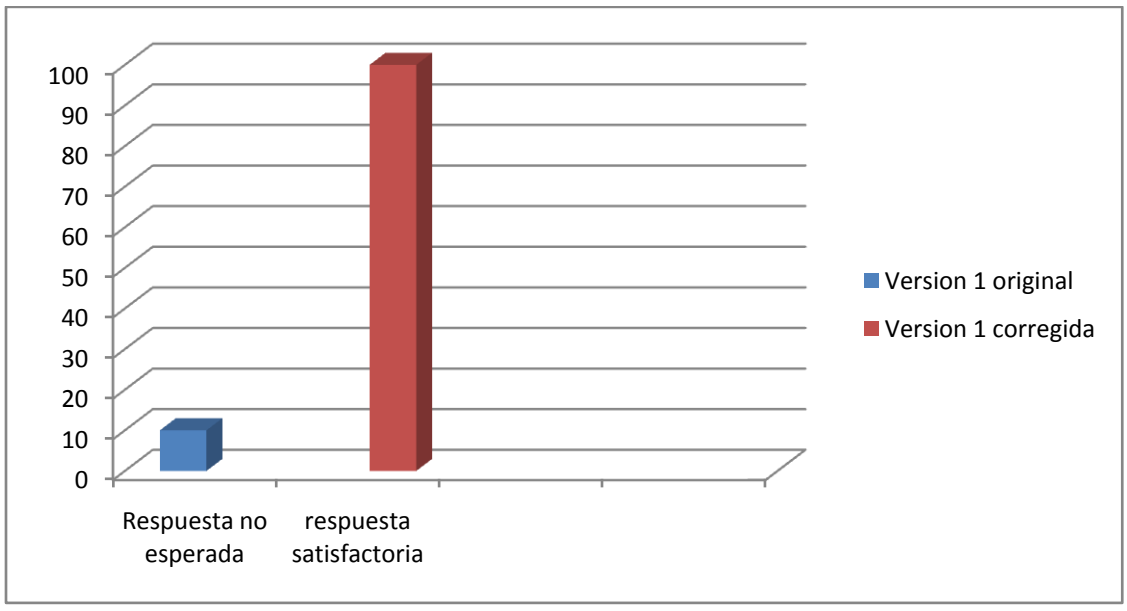


Figura 5.5 Respuesta de las pruebas realizadas de la versión 1

Una vez corregidos estos detalles de acceso (versión 2), las respuestas obtenidas fueron las esperadas, en cada uno de los casos. Tanto como los usuarios pudieron acceder como se pudieron comunicar, (ver figura 5.6).

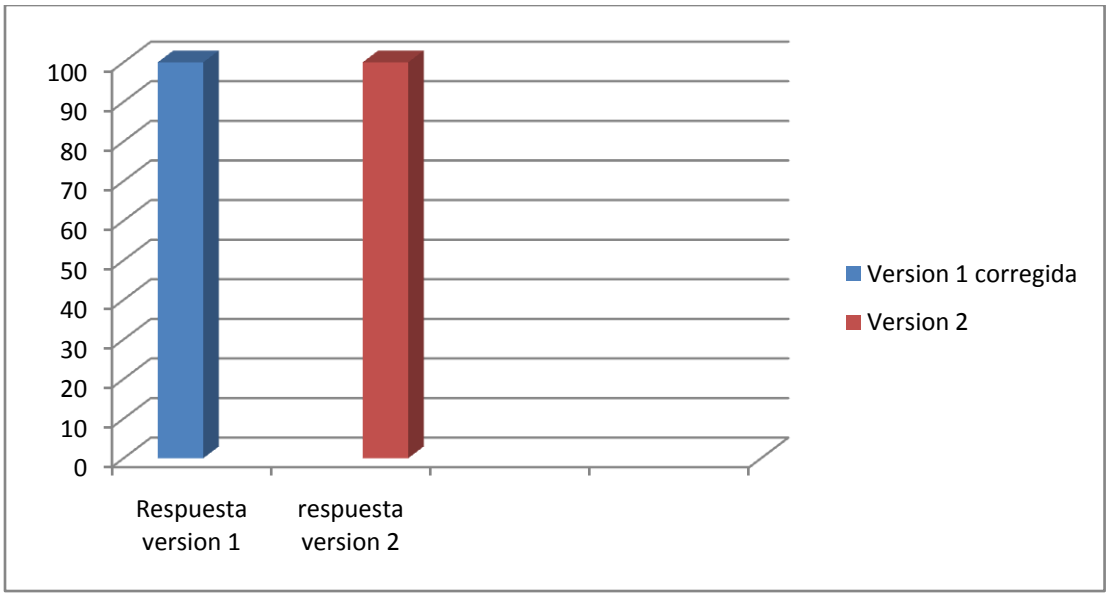


Figura 5.6 Respuesta de la versión 1 y 2 en las pruebas realizadas

Para las pruebas realizadas en cuanto a conexión, siempre se presentaron buenos resultados en cada uno de los sistemas operativos. El servidor respondió correctamente a las peticiones, dando acceso a cada uno de los usuarios conectados a la red y así, tener una buena comunicación entre toda la comunidad que hubiese iniciado sesión, (ver figura 5.7).

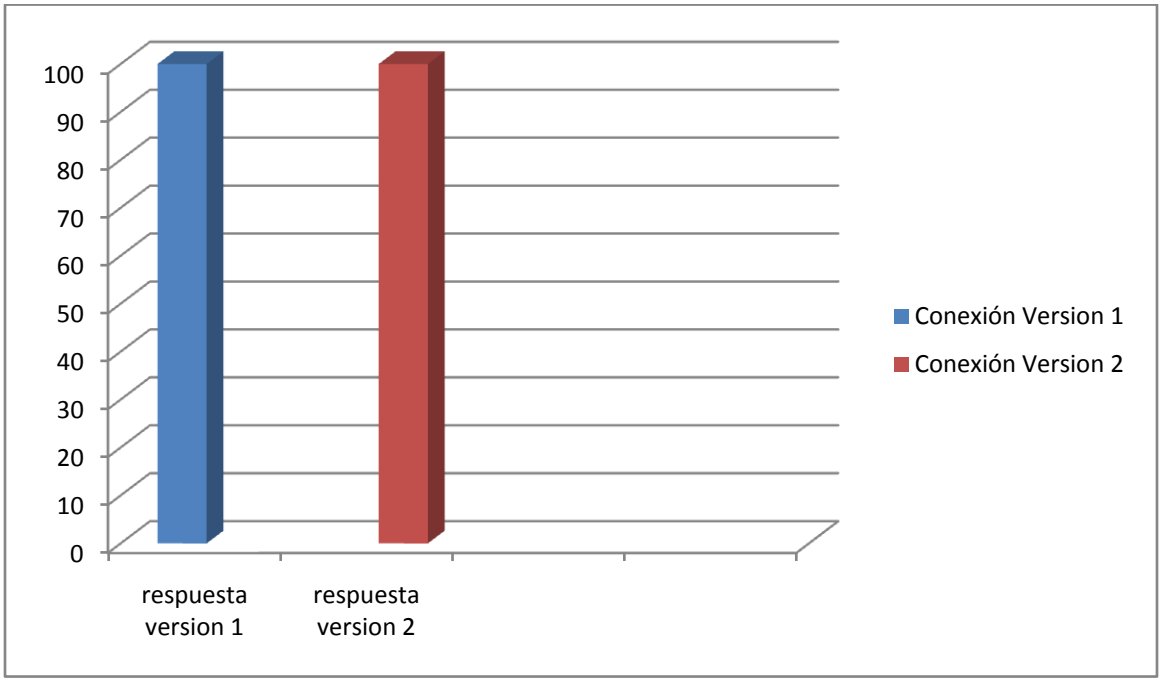


Figura 5.7 Respuesta a las pruebas de conexión

En el caso de la última versión, versión tres los mensajes ya pueden ser enviados a un usuario en específico y se limitaron los grupos a profesores y alumnos que lo conforman, para este caso todas las pruebas realizadas fueron satisfactorias, (ver figura 5.8).

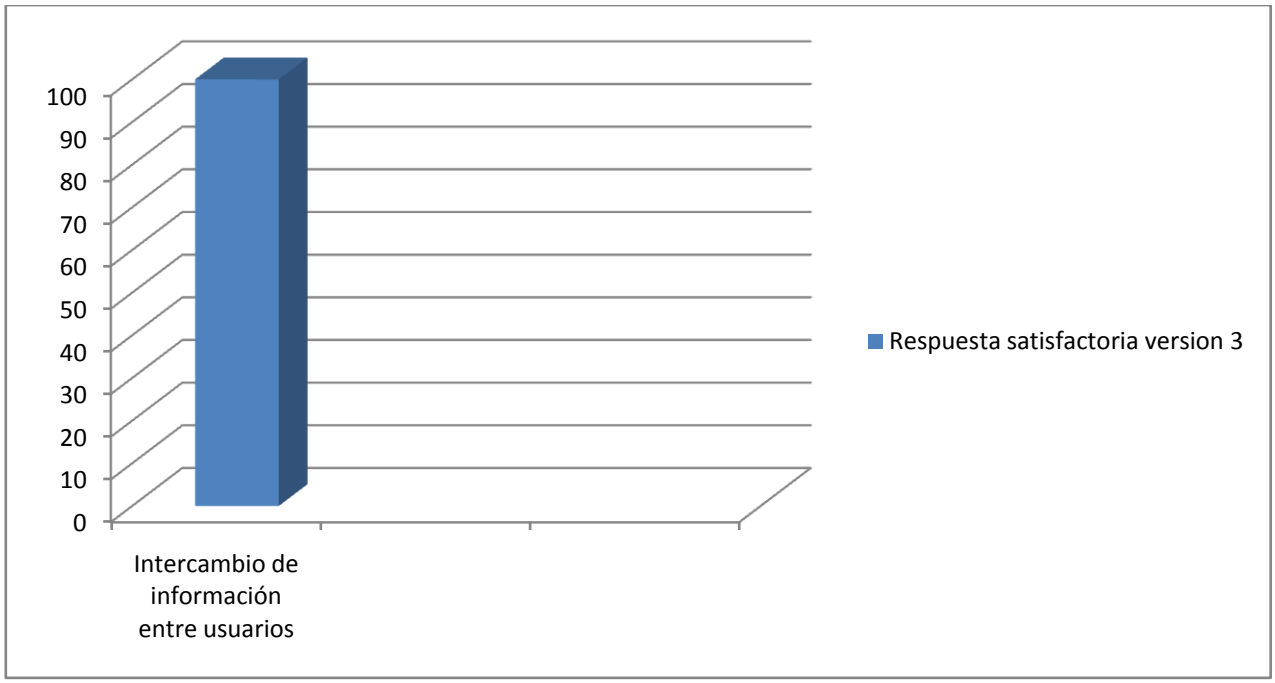


Figura 5.8 Respuesta a la versión 3

De manera General, a continuación se muestra el conjunto de las versiones generadas y su respuesta durante su desarrollo, (ver figura 5.9).

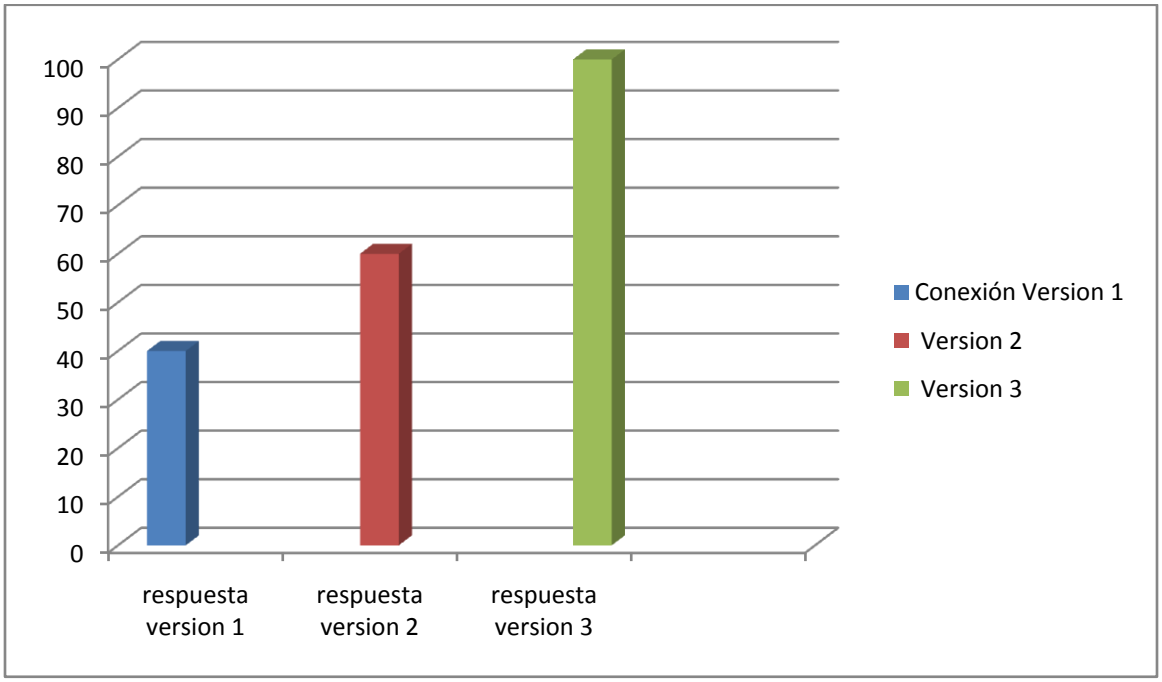


Figura 5.9 Comparación en respuesta a las pruebas realizadas en todas las versiones

Capítulo VI

Conclusiones y Recomendaciones

Dentro de este capítulo se analizaron las metas obtenidas después de haber realizado este trabajo. En esta parte, nos enfocaremos a todo el proceso que involucra nuestro proyecto y a todos los avances que hemos realizado y los resultados que hemos obtenido al realizar las pruebas ya mencionadas anteriormente.

CONCLUSIONES

Según los objetivos planteados. Se llegaron a las siguientes conclusiones.

Se desarrolló un software que ayuda con la comunicación entre usuarios, de esta manera se tendrá de una manera fácil y rápida para la resolución de dudas, entre otros tópicos.

También se desarrollo un sistema en el cual cada usuario posee de un nombre de usuario y una contraseña, esto fue desarrollado para que solo estudiantes que cuenten con permiso del administrador puedan acceder al sistema, y de esta manera se restrinja el acceso.

Otra de las cosas importantes que se lograron, fue el crear un ambiente sencillo y muy intuitivo, esto es que no es necesario que se capacite al usuario para que use el software. De esta manera se hace accesible para todo público, el cual tenga conocimientos básicos de computación.

Otro de los aspectos importantes a diferencia de otros mensajeros instantáneos, se observa en la seguridad, ya que no existen riesgos de transferencias de virus, de esta manera se ve más protegido nuestro equipo.

RECOMENDACIONES

Algunas de las recomendaciones para los alumnos de generaciones próximas es que continúen desarrollando este trabajo, de tal manera que con todas las herramientas ya expuestas, se les facilite su entendimiento y tengan una base en que apoyarse una vez que decidan continuar con el mismo.

Algunas de las mejoras para el sistema de mensajería instantánea son las que se mencionan a continuación:

- ***Encriptación***. Esta es una característica muy importante la cual es necesario implementarla, ya que si queremos tener una conversación más segura, es necesario implementar algún método de encriptación. Esta sugerencia esta dada, ya que durante el curso se estudiaron varias maneras en la cual personas ajenas a la red, pueden captar información sensible acerca de las conversaciones que se dieran. Así, si se tratara de una denuncia o de información sensible, estaría segura gracias a este sistema de seguridad.

- ***Gestor de archivos.*** Durante el desarrollo de nuestro software se observaron varios detalles faltantes, entre ellos un gestor de archivos, el cual nos permitirá enviar, recibir, entre otras cosas; así de esta manera estaremos dentro de una comunidad en donde se podrán compartir de manera fácil y rápida (esto depende de la velocidad de la conexión de cada usuario tenga).

Este gestor de archivos, ayudara en el objetivo de la interacción alumno-profesor, ya que mediante el uso de esta mejora es posible que el estudiante quien consulta puntos de vista, así como la revisión de trabajos, este podrá enviarlos directamente dentro de la comunicación del chat, para que el profesor que se encuentra del otro lado revise y envíe sus comentarios acerca del trabajo.

- ***Video conferencia***. Este es como la nueva generación de comunicación, así que este sería a lo mejor que podríamos llegar, ya que en vez de solo interactuar con texto plano ahora seria con video y voz. Los inconvenientes que tendría implementar este sistema de videoconferencia seria el ancho de banda para la conexión, ya que este tipo de servicios requiere gran demanda de recursos del sistema y de manera similar a la red.

- ***Interface.*** La interface seria uno de los aspectos que se mejorarían, dado que las tendencias en apariencias van cambiando, ahora se pueden ver programas con mejores apariencias, y estas son diseñadas dependiendo al cliente, para nuestro proyecto seria un poco más formal, ya que es orientado a la educación, pero sin perder también un toque fresco ya que también será usado por jóvenes.

- ***Software Libre.*** También se ha pensado que este software el cual fue creado para la comunidad estudiantil como al profesorado, podría ser en algún momento (parte del código) liberado, para que compañeros de la escuela mejoren, simplifiquen y den nuevas ideas acerca de las funciones y aspecto del mensajero. Esto se hace con la finalidad de que sirva también de apoyo y el proyecto tenga una continuación.

Estas son algunos de los puntos los cuales quedan abiertos a consideración para próximos desarrollos sobre el tema.

ANEXOS

Anexo [1] Glosario

- *Applet*. Son pequeños programas, que se pueden incluir en algunas páginas Web, con la finalidad de aportar más y mejores funcionalidades a dichas páginas.

- *Avatar*. En ciertos chats de la Internet, un avatar es una imagen que representa al usuario, con la misma función de un nick.

- Bytecode*.* Es un código intermedio más abstracto que el código máquina. Habitualmente se lo trata como a un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario que contiene código máquina producido por el compilador*.*

- *Datagrama*. Un datagrama es un fragmento de paquete que es enviado con la suficiente información como para que la red pueda simplemente encaminar el fragmento hacia el ordenador receptor, de manera independiente a los fragmentos restantes. Esto puede provocar una recomposición desordenada o incompleta del paquete en el ordenador destino.

- *Frame*. Opción que ofrece el lenguaje HTML de dividir una página web en varias zonas. Cada una de las cuales puede tener un contenido independiente de las demás de forma que cada zona es asimismo un frame. Un frame también se define la capa de enlace de datos que contiene la información de cabecera y cola que requiere una determinada red de comunicaciones.

- *Hashing* .Es un método para resumir o identificar un dato a través de la probabilidad, utilizando una función hash o algoritmo hash.

- *Heurística.* Capacidad de un sistema para realizar de forma inmediata innovaciones positivas para sus fines. La capacidad heurística es un rasgo característico de los humanos, desde cuyo punto de vista puede describirse como el arte y la ciencia del descubrimiento y de la invención o de resolver problemas mediante la creatividad y el pensamiento lateral o pensamiento divergente.

- *Html*. (Hipertext Markup Language) Lenguaje que permite escribir las páginas web a las que se accede a través de navegadores. Admite componentes hipertextuales (unidades de información asociadas para transmitir un mensaje) y multimedios (con imagen y sonido).

- *IP.*Internet Protocol. Permite enviar datos entre computadoras conectadas a una red.

- *IRC*. (Internet Relay Chat) es un protocolo de comunicación en tiempo real basada en texto, la cual permite debates en grupo y/o privado, el cual se desarrolla en canales de chat que generalmente comienzan con el carácter # o &, este último solo es utilizado en canales locales del servidor. Es un sistema de charlas muy popular actualmente y ampliamente utilizado por personas de todo el mundo.

- *Java*. Un lenguaje de programación para crear aplicaciones independientes de plataformas. Estos se introduce luego en el código html en páginas de Internet. Los programas Java se puede utilizar con cualquier sistema operativo y ha sido desarrollado por Sun Microsystems.

- *JavaScript*. Lenguaje de programación interpretado y diseñado para complementar las capacidades del HTML. El código de JavaScript es enviado al cliente como parte del código HTML de una página, y puede ser utilizado para lograr múltiples efectos especiales, como botones animados, sonido, etc.

- *JIT*. (Just In Time) Es un generador de código que convierte los bytecodes de Java en instrucciones de lenguaje de máquina. Algunas Máquinas Virtuales de Java, incluyendo la del navegador Netscape Navigator, traen un JIT junto con el intérprete de Java. Los programas en Java compilados con un JIT generalmente corren mucho más rápido que cuando el código es ejecutado por un intérprete.

- *Nick*. Palabra, nombre o pseudónimo que utiliza un usuario en un chat o un mensajero para identificarse y poder comunicarse.

- *Software*. Conjunto de programas que puede ejecutar una computadora. El software se divide en dos clases: software del sistema y software aplicativo o de aplicación.
    - S. de aplicación. Conjunto de programas escritos en cualquier lenguaje de programación que sirven para resolver, mediante una computadora, los problemas de una aplicación determinada
    - S. del sistema. Conjunto de programas que hacen que la computadora pueda ejecutar un programa escrito por el usuario. Entre ellos se encuentra el sistema operativo, los lenguajes, el ensamblador, etc.

- TCP. Transmission Control Protocol. Protocolo de control de la transmisión.

- UDP. User Datagram Protocol. Protocolo de Datagramas de Usuario.

Anexo [2] OCC

México-MEX-Ciudad de México - ANALISTA PROGRAMADOR EN JAVA

Giro de la empresa COMERCILIZADORA DE EQUIPO DE COMPUTO

Sueldo base programador de **Java** y Base de Datos $9000 + Bono por productividad $2000. Constante capacitación y plan de carrera.

Sueldo: $9,000 to $11,000 mensual
Puesto: Tiempo Completo
Código de Ref.: APJRH

Anexo [3] Código Fuente

```
/* Main.java*/

package mensajerover1;

import Interfaz.IniSesion;
import Interfaz.Ventana;

public class Main {

    public Main() {
    }

      /* Se crea un objeto tipo IniSesion  que es el objeto que contiene
      la ventana de inicio de sesión.*/
        public static void main(String[] args) {
        IniSesion Sesion = new IniSesion();
        Sesion.setTitle("Iniciar Sesión");
        Sesion.setVisible(true);
    }

}

/*IniSesion.java
*Esta clase genera la clase de inicio de sesión en la que se piden los
*datos del usuario para autentificarlo en la base de datos y darle acceso
*a la aplicación.
*/

package Interfaz;
import javax.swing.ImageIcon;

public class IniSesion extends javax.swing.JFrame {

    DataBase A = new DataBase();
    String usuario = null;
    String password = null;
    String ID = null;
    Contactos X;

    public IniSesion() {
        initComponents();
        this.setIconImage (new
ImageIcon(getClass().getResource("/imagenes/esime.gif")).getImage());
        getRootPane().setDefaultButton(Acepta);
    }
```

```
/* Código generado por NetBeans, en él se describe el diseño de la
*interfaz gráfica o los eventos generados en esta. */
         private void initComponents() {
        jLabel1 = new javax.swing.JLabel();
        user = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        error = new javax.swing.JLabel();
        pass = new javax.swing.JPasswordField();
        Acepta = new javax.swing.JButton();
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        jLabel1.setText("Usuario:");

        user.setText("Usuario");
        user.addFocusListener(new java.awt.event.FocusAdapter() {
            public void focusGained(java.awt.event.FocusEvent evt) {
                userFocusGained(evt);
            }
        });

        jLabel2.setText("Contrase\u00f1a");

        pass.setText("jPasswordField1");
        pass.addFocusListener(new java.awt.event.FocusAdapter() {
            public void focusGained(java.awt.event.FocusEvent evt) {
                passFocusGained(evt);
            }
        });

        Acepta.setText("Iniciar Sesi\u00f3n");
        Acepta.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                AceptaMouseClicked(evt);
            }
        });
        Acepta.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                AceptaActionPerformed(evt);
            }
        });
        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
DING)
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
DING)
                            .addComponent(error,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE, 256, Short.MAX_VALUE)
                            .addGroup(layout.createSequentialGroup()
                                .addGap(10, 10, 10)
                                .addComponent(user,
javax.swing.GroupLayout.DEFAULT_SIZE, 246, Short.MAX_VALUE))
                            .addComponent(jLabel1)
                            .addComponent(jLabel2)))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addGap(20, 20, 20)
                        .addComponent(pass,
javax.swing.GroupLayout.DEFAULT_SIZE, 246, Short.MAX_VALUE))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addContainerGap(171, Short.MAX_VALUE)
                        .addComponent(Acepta)))
                .addContainerGap())
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(error, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(user,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel2)
         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(pass, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(Acepta)
        .addContainerGap())
    );
    pack();
  }
```

```
/* Función que se encarga de autentificar al usuario dentro de la base de datos*/
   private void AceptaActionPerformed(java.awt.event.ActionEvent evt) {
      System.out.println("Autentificando al usuario,,,");
      usuario = user.getText();
      password = pass.getText();
       if(!usuario.equalsIgnoreCase("")&&!password.equalsIgnoreCase(""))
      {
         try{
            if(A.IniciarSesion(usuario,password))
            {
               System.out.println("Usuario autentificado");
               this.setVisible(false);
               X = new Contactos(usuario,password);
            }
            else
            {
               error.setText("Usuario o contraseña invalidos!!!");
            }
         }catch(Exception ex)
         {

         }
      }
       else
          error.setText("Debes ingresar tu Usuario y Contraseña!!!");
   }

   private void AceptaMouseClicked(java.awt.event.MouseEvent evt) {
      System.out.println("Autentificando al usuario,,,");
       if(!user.getText().equalsIgnoreCase("")&&!pass.getText().equalsIgnoreCase(""))
      {
         try{
            if(A.IniciarSesion(user.getText(),pass.getText()))
            {
               System.out.println("Usuario autentificado");
               this.setVisible(false);
               X = new Contactos(usuario,password);
            }
            else
            {
               error.setText("Usuario o contraseña invalidos!!!");
            }
         }catch(Exception ex)
         {

         }
      }
       Else
```

```
            error.setText("Debes ingresar tu Usuario y Contraseña!!!");
    }

   private void passFocusGained(java.awt.event.FocusEvent evt) {
       pass.selectAll();
    }
    private void userFocusGained(java.awt.event.FocusEvent evt) {
       user.selectAll();
    }
    public static void main(String args[]) {
       java.awt.EventQueue.invokeLater(new Runnable() {
          public void run() {
             new IniSesion().setVisible(true);
          }
       });
    }
    private javax.swing.JButton Acepta;
    private javax.swing.JLabel error;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPasswordField pass;
    private javax.swing.JTextField user;
}

/* Contactos.java*/
package Interfaz;
import java.awt.Color;
import java.util.Iterator;
import java.util.List;
import javax.swing.DefaultListModel;
import javax.swing.ImageIcon;
public class Contactos extends javax.swing.JFrame {
    DataBase db = new DataBase();
    DefaultListModel Contact = new DefaultListModel();
    List lista = null;
    static String usuario;
    static String password;
    static String grupo;
    static int ID;
/*Este es el constructor de la clase que se encarga de inicializar variables. */
    public Contactos(String usuario, String password) {
       initComponents();
       this.setIconImage (new
ImageIcon(getClass().getResource("/imagenes/esime.gif")).getImage());
       this.setBackground(java.awt.Color.BLACK);
       jLabel3.setIcon(new ImageIcon(getClass().getResource("/imagenes/ipnlogo.png")));
```

```
      jLabel4.setIcon(new ImageIcon(getClass().getResource("/imagenes/esimelogo.png")));
      this.repaint();
      this.setTitle("IPN - Escuela Superior de Ingenieria Mecanica y Electrica");
      Contactos.removeAll();
      this.usuario = usuario;
      this.password = password;
      Iniciar(usuario);
   }
/*    Esta función se encarga de obtener los datos del usuario y mostrarlos en pantalla. */
   public void Iniciar(String user)
   {
      lista = db.Contactos(ID=db.getID(user));
      Iterator it = lista.iterator();
      while (it.hasNext())
      {
         Contact.addElement((String)it.next());
      }
      grupo = db.getGrupo(user,ID);
      bienvenida.setText("Bienvenido "+usuario+",\n Grupo: "+grupo);
      Contactos.setModel(Contact);
      this.setVisible(true);
   }
/* Código generado por NetBeans, en él se describe el diseño de la
*interfaz gráfica o los eventos generados en esta. */
      jScrollPane1.setViewportView(Contactos);
      jLabel2.setText("Mensajero Instantaneo");
      jLabel1.setText("Contactos:");
      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
      getContentPane().setLayout(layout);
      layout.setHorizontalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addGroup(layout.createSequentialGroup()
                  .addGap(67, 67, 67)
                  .addComponent(jLabel2))
               .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                  .addContainerGap(13, Short.MAX_VALUE)
                  .addComponent(jLabel1)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
160, javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addGap(10, 10, 10))
               .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                  .addContainerGap()
```

```
      .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                  .addComponent(jLabel3, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.PREFERRED_SIZE, 38, javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addComponent(bienvenida, javax.swing.GroupLayout.DEFAULT_SIZE, 162,
Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 62,
javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addContainerGap())
      );
      layout.setVerticalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jLabel2)
            .addGap(14, 14, 14)
       .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE, 38,
Short.MAX_VALUE)
               .addComponent(jLabel4, javax.swing.GroupLayout.DEFAULT_SIZE, 38,
Short.MAX_VALUE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(bienvenida, javax.swing.GroupLayout.PREFERRED_SIZE, 13,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
      .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE)
               .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 437,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(37, 37, 37))
      );
      pack();
   }
   private void ContactosMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_ContactosMouseClicked
      if(evt.getClickCount()==2)
      {
         new Ventana((String)Contactos.getSelectedValue(),usuario,ID);
      }
   }
   public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
         public void run() {
            new Contactos(usuario,password).setVisible(true);
         }
      });
```

```
   private javax.swing.JList Contactos;
   private javax.swing.JLabel bienvenida;
   private javax.swing.JLabel jLabel1;
   private javax.swing.JLabel jLabel2;
   private javax.swing.JLabel jLabel3;
   private javax.swing.JLabel jLabel4;
   private javax.swing.JPanel jPanel1;
   private javax.swing.JScrollPane jScrollPane1;
   // Fin de declaración de variables
}

/* Ventana.java
*La clase ventana se encarga de generar la ventana mediante la cual se comunicarán los
*usuarios
*/

package Interfaz;
import java.io.*;
import java.net.*;
import javax.swing.ImageIcon;
public class Ventana extends javax.swing.JFrame {
   private static int PUERTO = 8000;
   private static String servidor = "192.168.1.68";
   protected Socket socketCliente;
   private Sesion sesion;
   private String user2;
   private String user1;
   private int ID;
   DataBase db = new DataBase();
   protected PrintStream salida;
   protected BufferedReader entrada;

  /*Se encuentra el constructor el cual se encarga de inicializar los datos  de los usuarios
*que van a entablar la comunicación, también se encarga de crear los sockets mediante el
cual los mensajes se enviarán al servidor*/
  public Ventana(String user2, String user1,int ID) {
       initComponents();
       getRootPane().setDefaultButton(BotonEnviar);
       Mensajes.setEditable(true);
       BotonEnviar.setEnabled(false);
       this.user2 = user2;
       this.user1 = user1;
       try {
          socketCliente = new Socket(servidor, PUERTO);
          salida = new PrintStream(socketCliente.getOutputStream());
          entrada = new BufferedReader(new
InputStreamReader(socketCliente.getInputStream()));
```

```
 sesion = new Sesion(this);
      this.setVisible(true);
      this.setTitle(user2 + " - Conversación");
      //salida.println(user1 + ID);
      System.out.println("Arrancando Cliente,,,");
   } catch (UnknownHostException e1) {
      System.out.println("Error - Servidor no encontrado - " + e1);
   } catch (IOException e2) {
      if (socketCliente != null) {
         try {
            socketCliente.close();
         } catch (IOException e3) {
         }
      }
      System.out.println("Error - Fallo al conectar al servidor - " + servidor + " - " + e2);
   }
}
protected void putMensajesTxt(String mensaje) {
   Mensajes.setText(mensaje);
}

protected String getMensajesTxt() {
   return Mensajes.getText();
}

public JTextArea getMensajes() {
   return Mensajes;
}

public void setMensajes(JTextArea Mensajes) {
   this.Mensajes = Mensajes;
}

public String getUltimoMsgTxt() {
   return ultimoMsg.getText();
}

public void setUltimoMsg(String ultimoMsgTxt) {
   this.ultimoMsg.setText(ultimoMsgTxt);
}
```

```
/* Código generado por NetBeans, en él se describe el diseño de la
*interfaz gráfica o los eventos generados en esta. */
    private void initComponents() {
      jScrollPane1 = new javax.swing.JScrollPane();
      Mensajes = new javax.swing.JTextArea();
      Texto = new javax.swing.JTextField();
      BotonEnviar = new javax.swing.JButton();
      setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
      setResizable(false);
      addWindowListener(new java.awt.event.WindowAdapter() {
         public void windowClosing(java.awt.event.WindowEvent evt) {
            formWindowClosing(evt);
         }
      });
      Mensajes.setColumns(20);
      Mensajes.setRows(5);
      jScrollPane1.setViewportView(Mensajes);
      Texto.addFocusListener(new java.awt.event.FocusAdapter() {
         public void focusGained(java.awt.event.FocusEvent evt) {
            TextoFocusGained(evt);
         }
      });
      Texto.addKeyListener(new java.awt.event.KeyAdapter() {
         public void keyPressed(java.awt.event.KeyEvent evt) {
            TextoKeyPressed(evt);
         }
         public void keyTyped(java.awt.event.KeyEvent evt) {
            TextoKeyTyped(evt);
         }
      });
      BotonEnviar.setText("Enviar");
      BotonEnviar.addMouseListener(new java.awt.event.MouseAdapter() {
         public void mouseClicked(java.awt.event.MouseEvent evt) {
            BotonEnviarMouseClicked(evt);
         }
      });
      BotonEnviar.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            BotonEnviarActionPerformed(evt);
         }      });
      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
      getContentPane().setLayout(layout);
      layout.setHorizontalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
            .addContainerGap()
```

```
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 511, Short.MAX_VALUE)
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                    .addComponent(Texto, javax.swing.GroupLayout.PREFERRED_SIZE, 432,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(BotonEnviar, javax.swing.GroupLayout.DEFAULT_SIZE, 73,
Short.MAX_VALUE)))
            .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 274,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(Texto, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(BotonEnviar))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        pack();
    }

    private void TextoKeyPressed(java.awt.event.KeyEvent evt) {
        if(Texto.getText()=="")
        {
            BotonEnviar.setEnabled(false);
        }else
        {
            BotonEnviar.setEnabled(true);
        }
    }
```

```
/*Se encarga de mandar el mensaje del usuario hacia el servidor */
   private void BotonEnviarActionPerformed(java.awt.event.ActionEvent evt) {
      if(BotonEnviar.isEnabled()==true)
      {
         if(!Texto.getText().equalsIgnoreCase(""))
         {
            salida.println("#"+user2+" "+Texto.getText());
Mensajes.setText(Mensajes.getText()+Texto.getText()+System.getProperty("line.separator
"));
            Texto.setText("");
            BotonEnviar.setEnabled(false);
         }
      }
   }
   private void formWindowClosing(java.awt.event.WindowEvent evt) {//GEN-
FIRST:event_formWindowClosing
      Mensajes.setText("Desconectando,,,");
   }
   private void BotonEnviarMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_BotonEnviarMouseClicked
       if(BotonEnviar.isEnabled()==true)
      {
         salida.println("#"+user2+" "+Texto.getText());
Mensajes.setText(Mensajes.getText()+Texto.getText()+System.getProperty("line.separator
"));
         Texto.setText("");
         BotonEnviar.setEnabled(false);
      }
   }
   private void TextoKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_TextoKeyTyped
      if(Texto.getText()=="")
      {
         BotonEnviar.setEnabled(false);
      }else
      {
         BotonEnviar.setEnabled(true);
      }
   }
   private void TextoFocusGained(java.awt.event.FocusEvent evt) {//GEN-
FIRST:event_TextoFocusGained
      BotonEnviar.setEnabled(true);
   }
```

```
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
            }
        });
    }
    });
    }
    private javax.swing.JButton BotonEnviar;
    private javax.swing.JTextArea Mensajes;
    private javax.swing.JTextField Texto;
    private javax.swing.JScrollPane jScrollPane1;
 }
/*La clase Sesion crea un hilo para recibir los mensajes que le llegan desde el servidor */
class Sesion extends Thread {

    private Ventana clienteChat;
    Calendar calendario;
    String anho, mes, dia, hora, minutos, segundos;

    public Sesion(Ventana clienteChat) throws IOException {
        this.clienteChat = clienteChat;
        start();
    }

    public void run() {
        String linea = null;
        try {
            while ((linea = clienteChat.entrada.readLine()) != null) {
                clienteChat.putMensajesTxt(clienteChat.getMensajesTxt() + linea +
System.getProperty("line.separator"));

clienteChat.getMensajes().setCaretPosition(clienteChat.getMensajes().getDocument().getLength());
                if (!linea.equals("****Bienvenido al Mensajero****")) {
                    calendario = new GregorianCalendar();
                    anho = ""+calendario.get(Calendar.YEAR);
                    if(calendario.get(Calendar.MONTH)<10)
                        mes = "0"+calendario.get(Calendar.MONTH);
                    else
                        mes = ""+calendario.get(Calendar.MONTH);
                    if(calendario.get(Calendar.DAY_OF_MONTH)<10)
                        dia = "0"+calendario.get(Calendar.DAY_OF_MONTH);
                    else
                        dia = ""+calendario.get(Calendar.DAY_OF_MONTH);
                    if(calendario.get(Calendar.HOUR_OF_DAY)<10)
                        hora = "0"+calendario.get(Calendar.HOUR_OF_DAY);
                    else
                        hora = ""+calendario.get(Calendar.HOUR_OF_DAY);
                    if(calendario.get(Calendar.MINUTE)<10)
                        minutos = "0"+calendario.get(Calendar.MINUTE);
                    else
                        minutos = ""+calendario.get(Calendar.MINUTE);
```

```
                if(calendario.get(Calendar.SECOND)<10)
                   segundos = "0"+calendario.get(Calendar.SECOND);
                else
                   segundos = ""+calendario.get(Calendar.SECOND);
                clienteChat.setUltimoMsg("Ultimo mensaje recibido el día "+ dia + "/" + mes + "/" + anho
+" a las " + hora + ":" + minutos + ":" + segundos );
                repSound();
             }
             if (linea.equals("Adios Lumpensin,,,") || linea.equals("> Conexion cerrrada por
inactividad,,,")) {
                try {
                   Thread.sleep(5000);
                } catch (InterruptedException e1) {
                }
                break;
             }
          }
       } catch (IOException e2) {
          e2.printStackTrace();
       } finally {
          try {
             clienteChat.dispose();
             clienteChat.entrada.close();
             clienteChat.salida.close();
             clienteChat.socketCliente.close();
          } catch (IOException e3) {
          }
          System.exit(0);
       }
    }

    public void repSound() {
       Player player;
       try {
          FileInputStream fis = new FileInputStream("C:" + File.separatorChar + "lol" +
File.separatorChar + "mess.mp3");
          player = new Player(fis);
          player.play();
       } catch (JavaLayerException ex) {
          Logger.getLogger(Sesion.class.getName()).log(Level.SEVERE, null, ex);
       } catch (IOException ioe) {
          ioe.printStackTrace();
       }
    }
}
```

```
 /*DataBase.java
*  Clase se encarga de realizar la conexión con la base de datos para obtener la
*información que se requiere del usuario*/

package Interfaz;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import vo.Usuario;

public class DataBase {

   public DataBase() {
   }
   static String bd = "messenger";
   static String login = "messenger";
   static String password = "messenger";
   static String url = "jdbc:mysql://localhost/" + bd;

   public Usuario IniciarSesion(String usr, String pwd) throws Exception {
      System.out.println("Entrando a DB");
      Connection conn = null;
      try {
         Class.forName("com.mysql.jdbc.Driver").newInstance();
         conn = DriverManager.getConnection(url, login, password);
         ResultSet rs = null;
         String PASSWORD = null;
         System.out.println("Intentando conectar a la base,,,");
         if (conn != null) {
            System.out.println("Conexion con la DB establecida con exito,,,");
            PreparedStatement A = conn.prepareStatement("SELECT id, usuario, password,
grupo FROM usuarios WHERE usuario=?");
            A.setString(1, usr);
            rs = A.executeQuery();
            while (rs.next()) {
               Usuario user = new Usuario();
               int i = 1;
               user.setId(rs.getInt(i++));
               user.setUsuario(rs.getString(i++));
               user.setPassword(rs.getString(i++));
               user.setGrupo(rs.getString(i++));
               if (user.getPassword().equals(pwd)) {
                  return user;
               }
            }
         }
```

```
      } catch (SQLException ex) {
         System.out.println("Hubo un problema al intentar conectarse con la base de datos
" + url);
         System.out.println(ex.getErrorCode());
      } catch (ClassNotFoundException ex) {
         System.out.println(ex);
      }
      return null;
   }

   public String getID(String user) {
      System.out.println("Entrando a getID");
      Connection conn = null;
      String ID = null;
      try {
         Class.forName("com.mysql.jdbc.Driver").newInstance();

         conn = DriverManager.getConnection(url, login, password);
         ResultSet rs = null;
         String PASSWORD = null;
         System.out.println("Intentando conectar a la base,,,");
         if (conn != null) {
            System.out.println("Conexion con la DB establecida con exito,,,");
            PreparedStatement A = conn.prepareStatement("select ID from usuarios where
user=?");
            A.setString(1, user);
            rs = A.executeQuery();
            while (rs.next()) {
               ID = rs.getString(1);
            }
         }
      } catch (ClassNotFoundException ex) {
         ex.printStackTrace();
      } catch (InstantiationException ex) {
         ex.printStackTrace();
      } catch (IllegalAccessException ex) {
         ex.printStackTrace();
      } catch (SQLException ex) {
         System.out.println("Hubo un problema al intentar conectarse con la base de datos
" + url);
         System.out.println(ex.getErrorCode());
      }
      return ID;
   }
```

```
   public List getContactos(String grupo) {
      System.out.println("grupo"+grupo);
      Connection conn = null;
      List<Usuario> ListaContactos = new ArrayList<Usuario>();
      try {
         Class.forName("com.mysql.jdbc.Driver").newInstance();

         conn = DriverManager.getConnection(url, login, password);
         ResultSet rs = null;
         Usuario user =null;
         System.out.println("Intentando conectar a la base,,,");
         if (conn != null) {
            PreparedStatement A = conn.prepareStatement("SELECT id, usuario, password,
grupo FROM usuarios WHERE grupo=?");
            A.setString(1, grupo);
            System.out.println("o.O!!!!!!!!!!!!!!!!!!!");
            rs = A.executeQuery();
            while (rs.next()) {
               user = new Usuario();
               user.setId(rs.getInt("id"));
               user.setUsuario(rs.getString("usuario"));
               user.setPassword(rs.getString("password"));
               user.setGrupo(rs.getString("grupo"));
               ListaContactos.add(user);
            }
         }
      } catch (ClassNotFoundException ex) {
         ex.printStackTrace();
      } catch (InstantiationException ex) {
         ex.printStackTrace();
      } catch (IllegalAccessException ex) {
         ex.printStackTrace();
      } catch (SQLException ex) {
         System.out.println("Hubo un problema al intentar conectarse con la base de datos
" + url);
         System.out.println(ex.getErrorCode());
      }
      return ListaContactos;
   }
}
```

```
/* Servidor.java
*Se encarga de esperar una conexión de los usuarios, de generar un hilo para cada uno de
*ellos  y de recibir y manejar cada uno de los mensajes.*/

package servidor;
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.net.*;
public class Servidor {
   ServerSocket Servidor;
   Socket Cliente;
   private final static int PUERTO = 8000;
   protected static final int TimeOut = 0;
   public static void main(String[] args){
      Servidor Server = new Servidor();
      Server.Arranca();
      Server.Clientes();
   }

   private void Arranca(){
      try{
         Servidor = new ServerSocket(PUERTO);
         System.out.println("Servidor en marcha,,,");
      } catch(BindException e1){
         System.out.println("Error - "+e1);
         System.exit(-1);
      } catch(SecurityException e2){
         System.out.println("Error - "+e2);
         System.exit(-1);
      } catch(IOException e3){
         System.out.println("Error -"+e3);
         System.exit(-1);
      }
   }
   private void Clientes(){
      Cliente = null;{
         while(true){
            try{
               Cliente = Servidor.accept();
               try{
                  new HiloCliente(Cliente);
               } catch(IOException e1){
                  try{
                     Cliente.close();
                  }catch(IOException e2){
                  }
               }
```

```
            } catch(SecurityException e3){
               if(Servidor != null){
                  try{
                     Servidor.close();
                  } catch(IOException e4){
                  }
               }
               System.out.println("Error - "+e3);
            } catch(IOException e5){
            }
         }
      }
   }
}

/*HiloCliente se encarga de manejar los mensajes de cada usuario*/

  class HiloCliente extends Thread{
   private String Nick;
   private static List Clientes = new ArrayList();
   private static List Usuarios = new ArrayList();
   private Socket socket;
   private BufferedReader entrada;
   private PrintWriter salida;

   HiloCliente(Socket socket)throws IOException {
      this.socket = socket;
      entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      salida = new PrintWriter(socket.getOutputStream(),true);
      start();
   }
   /*Sobrecarga del metodo run();*/
   public void run(){
      String mensaje="";
      try {
         Nick=(entrada.readLine());
         Iterator it = Clientes.iterator();
         Conectar(this);
         salida.println("> Bienvenido "+Nick.substring(0,Nick.length()-2));
         socket.setSoTimeout(Servidor.TimeOut);
         while((mensaje=entrada.readLine())!=null){
            System.out.println(Nick + mensaje);
            if((mensaje.equals("/desconectar"))){
               salida.println("Hasta luego,,,");
               break;
```

```
            } else if(mensaje.startsWith("#")){
               String palabras[]=mensaje.split(" ");
               Iterator itera = Clientes.iterator();
               while(itera.hasNext()){
                  HiloCliente temp = (HiloCliente)itera.next();
                  String nick = palabras[0].substring(1,palabras[0].length());
                  if(temp.Nick.equalsIgnoreCase(nick)) {
                     String msj = this.Nick.substring(0,this.Nick.length()-2) + " dice: " +
mensaje.substring(palabras[0].length(),mensaje.length());
                     escribeUsuario(temp,">>> "+msj);
                  }
               }

            } else if((mensaje.equals("/listar"))){
               escribeUsuario(this,"> "+listaUsuarios());
            } else{
               MandaMensaje(Nick+"-> "+mensaje);
            }
         }
      } catch(InterruptedIOException e1){
         escribeUsuario(this,"> " +"Conexion cerrrada por inactividad,,,");
      } catch(IOException e2){
      } finally{
         Desconectar(this);
         limpiar();
      }
   }
   private void limpiar(){
      if(entrada!=null){
         try{
            entrada.close();
         } catch(IOException e1){
         }
         entrada = null;
      }
      if(salida != null){
         salida.close();
         salida = null;
      }
      if(socket != null){
         try{
            socket.close();
         } catch(IOException e2){
         }
         socket = null;
      }
   }
```

```
   /*Se encarga de eliminar un hilo de un cliente que ya no está desconectado*/
   private static synchronized void Desconectar(HiloCliente Hilo){
      Clientes.remove(Hilo);
   }

   /*Crea un nuevo hilo para un nuevo usuario conectado*/
   private static synchronized void Conectar(HiloCliente Hilo){
      Clientes.add(Hilo);
   }

  /*Manda un mensaje a todos los usuarios conectados*/
   private synchronized void MandaMensaje(String msj){
      Iterator it = Clientes.iterator();
      while(it.hasNext()){
         HiloCliente temp = (HiloCliente)it.next();
         if(!(temp.equals(this))) {
            escribeUsuario(temp,msj);
            System.out.println(msj);
         }
      }
   }

  /*Manda un mensaje a un solo usuario*/
   private synchronized void escribeUsuario(HiloCliente hilo, String txt){
      (hilo.salida).println(txt);
   }

   /*Obtiene la lista de los usuarios conectados*(
   private static synchronized StringBuffer listaUsuarios(){
      StringBuffer cadena = new StringBuffer();
      for(int i=0; i<Clientes.size();i++){
         HiloCliente temp=(HiloCliente)(Clientes.get(i));
         cadena.append((((HiloCliente)Clientes.get(i)).Nick)).append("|<->|");
      }
      return cadena;
   }

}
```

Bibliografía

[1] Ceballos Sierra, Francisco Javier. Java™ 2 *Interfaces Gráficas y Aplicaciones para Internet.* México, Editorial Alfaomega.

[2] Morales, C.; Turcott, V.; Campos, A.; Lignan, L. (1998). *Actitudes de los escolares hacia la computadora y los medios para el aprendizaje*. México. ILCE.

[3] Morales, C.; González, I.; Carmona, V.; Soto, C.; Torres, K.; López, O.; Zárate, M. (2000). *Ambientes de Aprendizaje Computarizado*. México. ILCE.

[4] http://es.geocities.com/chpruneda/vol11hpciinnovacion.pdf

[5] http://www.netmarketing.com.mx/estadisticas-mexico.html

[6] http://www.pilarteens.com.ar/modules.php?name=News&file=article&sid=15

[7] http://www.forospyware.com/t64782.html

[8] http://pcdoctor.com.mx/Radio%20Formula/temas/Messengers.htm