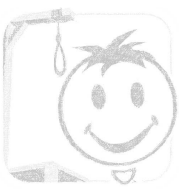
Rubén Arcos Ortega

2º C.F.G.S. Desarrollo de aplicaciones multiplataforma

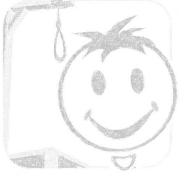
Servicios y procesos

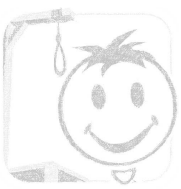
AHORCADO CON CHAT

Juego del ahorcado con implementación en red



Índice

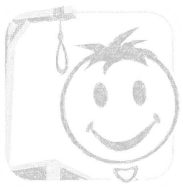


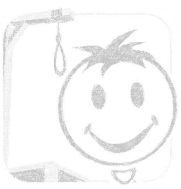


En la siguiente documentación, se detalla la práctica evaluable relacionada con la asignatura de Servicios y procesos, con fecha de entrega del domingo 24 de enero de 2015.

A continuación se encuentra el planteamiento del problema a resolver, la estructuración y motivación del proyecto, junto con el diagrama de clases del mismo y los códigos fuentes relacionados.

He de dejar constancia que el proyecto ha supuesto una gran puesta en práctica de los conocimientos adquiridos durante el periodo de la asignatura en el ámbito de la programación de comunicaciones en red (*sockets)* mediante el lenguaje de programación JAVA, dando paso y una nueva visión de las comunicaciones e interactuación entre aplicaciones clientes y alojadas en servidores, como en los inconvenientes que con lleva y las tecnologías a conocer.





1 Motivación del proyecto

Realización de una aplicación mediante el lenguaje de programación JAVA, que incorpore y requiera en la medida más cercana a la realidad del establecimiento de comunicación entre un servidor y varios clientes, con similitud a un problema que requiera de las mismas características. Utilización de entorno gráfico basado en *Java Swing* y modularidad de la aplicación basado en patrón de funcionalidad frente a la comunicación.

2 Planteamiento del problema real

2.1 Entorno

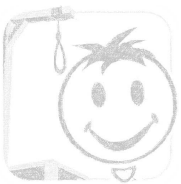
Sistema de control del juego del ahorcado, gestionando las conexiones, límites de usuarios, comunicaciones y consulta de aciertos y errores desde el servidor. Y teniendo la gestión de la interactuación con el usuario y los envíos y recepciones de los datos de la partida.

2.2 Situación real y restricciones habituales

2.2.1 El juego

Consiste en la adivinación por parte de los jugadores de la palabra elegida por uno de los miembros que no juega, esta puede ser mediante letras sueltas o por la palabra al completo. En ambos casos se acumulan fallos, los cuales se van representando en forma de un personaje que se va completando hasta estar completo y colgar de una horca.

Finalmente gana el jugador que acierte todas las letras o la palabra en cualquier momento.



2.2.2 Los jugadores

El número de jugadores mínimo que se requiere es de dos y máximo el número de letras de la palabra, para que al menos todos puedan tener una posibilidad.

2.3 Restricciones y condiciones aplicadas en la aplicación

- Para que se dé el comienzo de la partida debe haber un mínimo de 2 jugadores y un máximo delimitado mediante código por la constante `MAX_CONEXIONES` que se encuentra actualmente en 4.
- La partida se encontrará a la espera de conexiones al iniciar el servidor.
- La elección de la palabra se realizará en el servidor, bien de forma manual o aleatoria. Esto implicará la comunicación a los usuarios de la elección de la palabra, y la finalización de la partida en caso de que se encontrase en curso.
- Una vez se confirme la palabra, comienza la partida, y si había alguna anterior se eliminarán sus datos y estadísticas
- No se realizará distinción entre las palabras a resolver y letras sensibles a mayúsculas, minúsculas o acentuadas. Todos los casos se tomarán como válidos.
- No se contabilizará como error el intento de consulta de una letra ya resuelta o fallida.
- Si se contempla como fallo la errónea resolución de la palabra completa.
- Independientemente al estado de la partida en curso o la existencia de la misma, siempre se encuentran disponibles en todo momento las funcionalidades de comunicación de carácter bidireccional cliente-servidor y con extensión a todo el grupo de usuarios conectados.
- Todos los mensajes internos de carácter técnico quedan ocultos al usuario.
- Al comienzo del juego se asignan los turnos de usuarios, en orden estricto de conexión.
- El turno transcurrirá en orden de conexión cuando bien se produzca un fallo en la consulta de una letra o en la resolución errónea de la palabra, una vez terminada la ronda comienza de nuevo desde el principio.
- La contabilización de 7 fallos conllevará el fin de la partida para ese jugador.

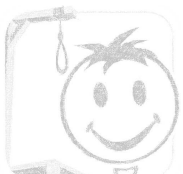
- Gana el jugador que resuelva la palabra tanto por letras como por adivinación de ésta.
- No se puede retirar/desconectar de la partida un usuario que se encuentre con la jugada en curso. Si mientras juegan los demás, excepto cuando sea el único usuario de la partida.
- Los usuarios no podrán unirse a la partida una vez empezada, el intento de conexión será rechazado desde el servidor.
- Los datos de turno y usuarios conectados son comunicados y actualizados a todos los usuarios, inclusive el servidor en todo momento y a tiempo real.
- Cuando un jugador gana la partida, se permite a todos que no se desconecten, mediante una consulta, y se reanudará una nueva partida con una nueva palabra previo cambio en el servidor. Durante este periodo sí se podrán realizar nuevas conexiones para unirse a la partida.
- El servidor podrá desconectar de forma controlada a todos los usuarios, sin su autorización, aunque con previo aviso y con la consiguiente finalización de la partida y cierre del canal de comunicación general.
- Se establece un protocolo propio de comunicaciones tanto del lado del servidor como del cliente.

3 Análisis y diseño del problema

3.1 Estructura y lógica de la aplicación

3.1.1 Proyecto servidor `[Servidor.java]`

- Contiene el inicio del flujo de ejecución de la aplicación.
- Gestiona la aceptación de las comunicaciones de los usuarios por un puerto externo de comunicaciones, y creando un hilo de proceso independiente para cada uno de ellos, por el cual establecerá la comunicación por un puerto local específico.



- Gestión y comprobación de las consulta de letras contenidas en la palabra a resolver.
- Gestión y comprobación de la consulta de la palabra completa a resolver.
- Gestión de turnos de los jugadores, cambia con fallos y desconexiones.
- Gestión de inicio, suspensión, parada y finalización de la partida con control de usuarios activo y estados.

3.1.2 Proyecto servidor `[ServidorConexionUsuario.java]`

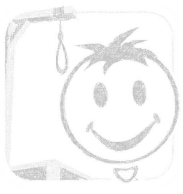
- Recepción y envío de mensajes tanto grupales como privados.
- Recepción y envío de objetos serializados.
  - Con los datos de la partida: consulta de letra, palabra actualmente en resolución (con las incógnitas, no completa), turno de jugador en curso…
  - Lista de usuarios conectados para la comunicación grupal.
- Gestión de desconexión de los usuarios, recibe la petición y el servidor cierra previamente la comunicación con el solicitante y los procesos relacionados. Una vez se ha producido la desconexión se comunica al cliente para que cierre su aplicación automáticamente.
- Gestión del protocolo de comunicaciones.

3.1.3 Proyecto servidor, utilidades `[LeerPalabrasFichero.java]`

- Extrae las palabras almacenadas en el fichero de texto `listadoPalabras.txt` separadas con salto de línea `CR+LF` en codificación `UTF-8` para su lectura mediante la clase `FileReader` de java y su posterior conversión a una estructura compatible con el elemento `JComboBox` en la cual será utilizada.

3.1.4 Proyecto servidor y cliente, utilidades `[Utils.java]`

- Gestiona la utilización de la hora actual del sistema para los registros de la aplicación y la anulación de caracteres especiales al estándar de texto, como



tildes y diferencias entra mayúsculas y minúsculas para la comprobación de palabras.

3.1.5 Proyecto servidor, interfaz gráfica `[Ventana.java]`

- Gestiona la interfaz gráfica con el usuario y la gestión de los controles. Así como el estado de comunicación con el servidor.

3.1.6 Proyecto cliente `[Cliente.java]`

- Contiene el inicio del flujo de ejecución de la aplicación.
- Gestiona las acciones de los elementos de interactividad con el usuario en el entorno gráfico.
- Gestiona y controla el envío de mensajes del usuario y recepción del servidor.
- Gestiona y controla la desconexión del usuario, tanto de la partida como del servidor.
- Gestiona el envío de la consulta de las letras y palabra a adivinar.
- Comprueba la validación de la entrada de datos del usuario.
- Informa al usuario del estado y restricciones de la partida.
- Solicita la entrada de datos de conexión y de la partida al usuario.
- Realizar la lectura de la configuración con los datos de conexión de usuario en el fichero de texto, a través de las utilidades del proyecto (se detallará más adelante).
- Recepción y envío de mensajes tanto grupales como privados.
- Recepción y envío de objetos serializados.
    - Con los datos de la partida: consulta de letra, palabra actualmente en resolución (con las incógnitas, no completa), turno de jugador en curso…
    - Lista de usuarios conectados para la comunicación grupal.
- Gestión del protocolo de comunicaciones.
- Contabilización de los fallos y aciertos del usuario.
- Desconexión controlada de la comunicación.
- Desactivación/activación de los controles necesarios según el estado de la partida y las acciones anteriores (letras acertada, fallidas y estado turno).

- Actualización de los datos de la partida a tiempo real y conexiones de los usuarios de comunicaciones. También de los cambios en la interfaz gráfica.
- Gestión del ganador y perdedor de la partida.

3.1.7 Proyecto cliente, interfaz gráfica `[Ventana.java]`

- Gestiona la interfaz gráfica con el usuario y la gestión de los controles. Así como el estado de comunicación con el servidor.

3.1.8 Proyecto servidor, interfaz gráfica `[Ventana.java]`

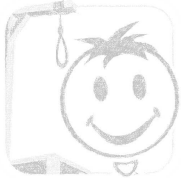
- Gestiona la interfaz gráfica con el usuario y la gestión de los controles. Así como el estado de comunicación con el servidor.

3.1.9 Proyecto cliente, utilidades `[LeerDatosConexion.java]`

- Extrae las palabras almacenadas en el fichero de texto `datosConexion.txt` separadas con salto de línea `CR+LF` en codificación `UTF-8` para su lectura mediante la clase `FileReader` de java y su posterior conversión a una estructura propia de datos en la clase `DatosConexion`.

3.1.10 Proyecto cliente y servidor, almacenamiento de la información en formato propio `[DatosPartida.java y DatosConexion.java]`

- Contiene la estructura necesaria para la lectura/escritura y acceso/validación de los datos necesarios para las conexiones y la gestión de la partida.



3.2 Protocolo de comunicación

Debido a las características de la aplicación, he considerado la mejora forma de resolverlas comunicaciones entre usuarios y el servidor, mediante la creación de un protocolo propio, el cual transmitirá información en algunos casos y en otros la advertencia del envío de un objeto serializado a continuación, en lugar de texto.

Para la gestión del protocolo he establecido unas cadenas de texto específicas para cada acción, que serán enviadas y recibidas, tanto en cliente como en servidor.

En el lado del servidor son gestionadas por la clase `ServidorConexionUsuario`, en el método `gestionarComunicación()`, e igualmente en la clase `Cliente`, en el método `GestionarMensajes()` del lado del cliente. El protocolo queda así:

- `[DESCONECTAR]`: El cliente envía este comando para la petición de desconexión al servidor. En el servidor es recibido y cerrado el canal de comunicación, socket e hilo de proceso con este usuarios. Posteriormente, el servidor lo envía al cliente, justo antes de desconectar, para que cuando en el cliente es recibido se realice el cierre del canal de comunicación y del socket activo.
- `[OBJETO]`: advierte que a continuación se va a realizar en envío de un objeto, el cual se encuentra serializado, para que se compare con los conocidos y se convierta a uno de ellos. También requiere del cambio de escucha de flujo de datos. Lo mismo se realiza pero en proceso inverso antes del envío de un objeto. Este tipo de comunicación se utiliza en todas las actualizaciones de los datos de la partida con la clase `DatosPartida` y con la lista de usuarios conectados con un objeto de tipo `String[]`.
- `[INICIAR PARTIDA]`: indica que se cerrarán las conexiones con nuevos usuarios, que el primer jugador que se conectó va a empezar la partida y que se recibirá un objeto con los datos de la misma.
- `[NOMBRE]`: a continuación recibe una cadena de texto con el nombre del usuario que se acaba de conectar para su almacenamiento.
- `[PARAR PARTIDA]`: cambio de palabra o suspensión temporal de la partida.
- `[PARAR PARTIDA FIN]`: un usuario ha adivinado la partida. Se comunica la posibilidad de desconexión a los usuarios.
- `[LETRA FALLIDA]`: el usuario que la recibe, se envía por privado, contabilizará un fallo y perderá el turno.
- `[LETRA ACERTADA]`: el usuario que la recibe, se envía por privado, contabilizará un acierto.

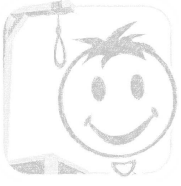
El resto de operaciones de la aplicación de comunicación, se realiza mediante la comprobación de los datos almacenados en los objetos de envío/recepción serializados.

3.3 Diagrama de clases

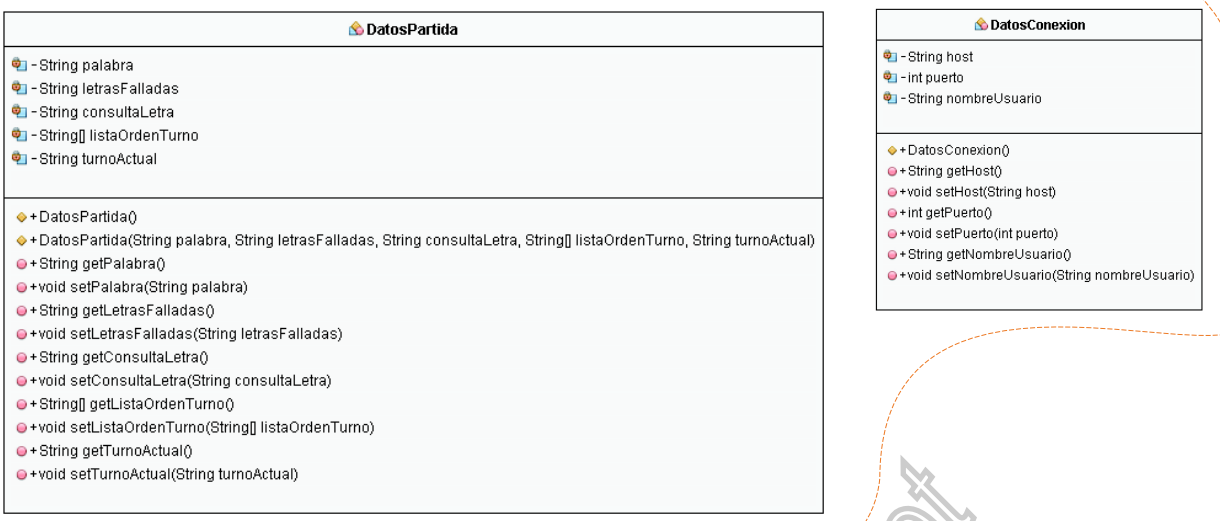
Gestión **cliente** – aplicación de conexión de usuario

Cliente – Lógica y control

Cliente GUI – Interfaz gráfica

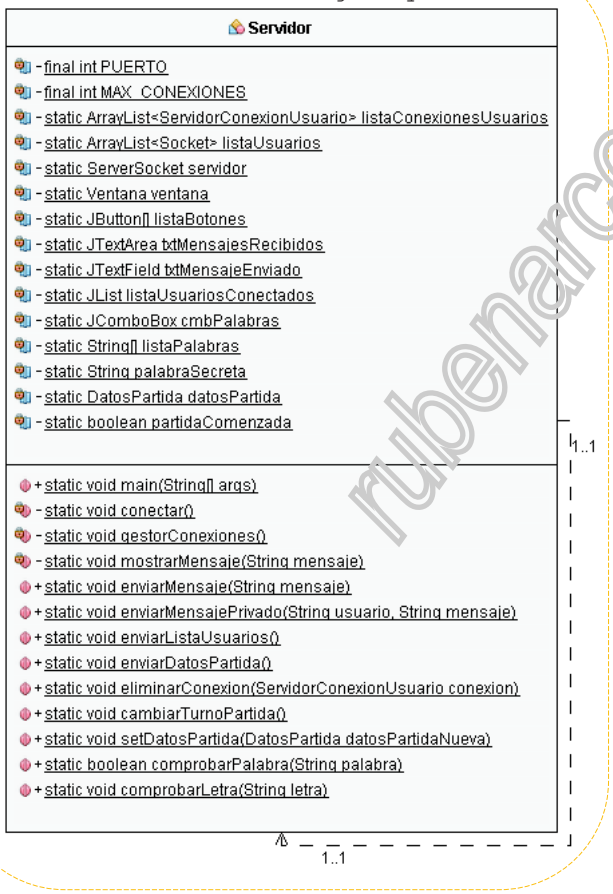


Almacenamiento de información – Objetos propios

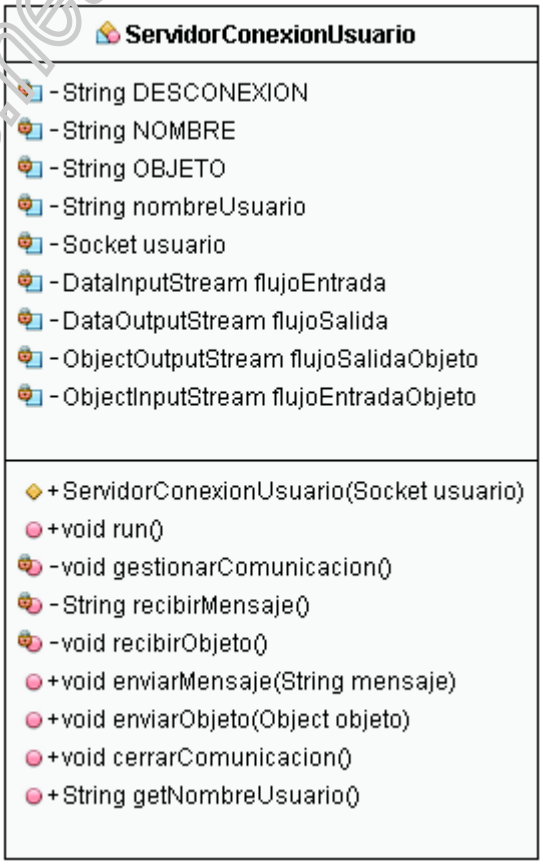


Gestión **servidor** – aplicación de conexión de usuario

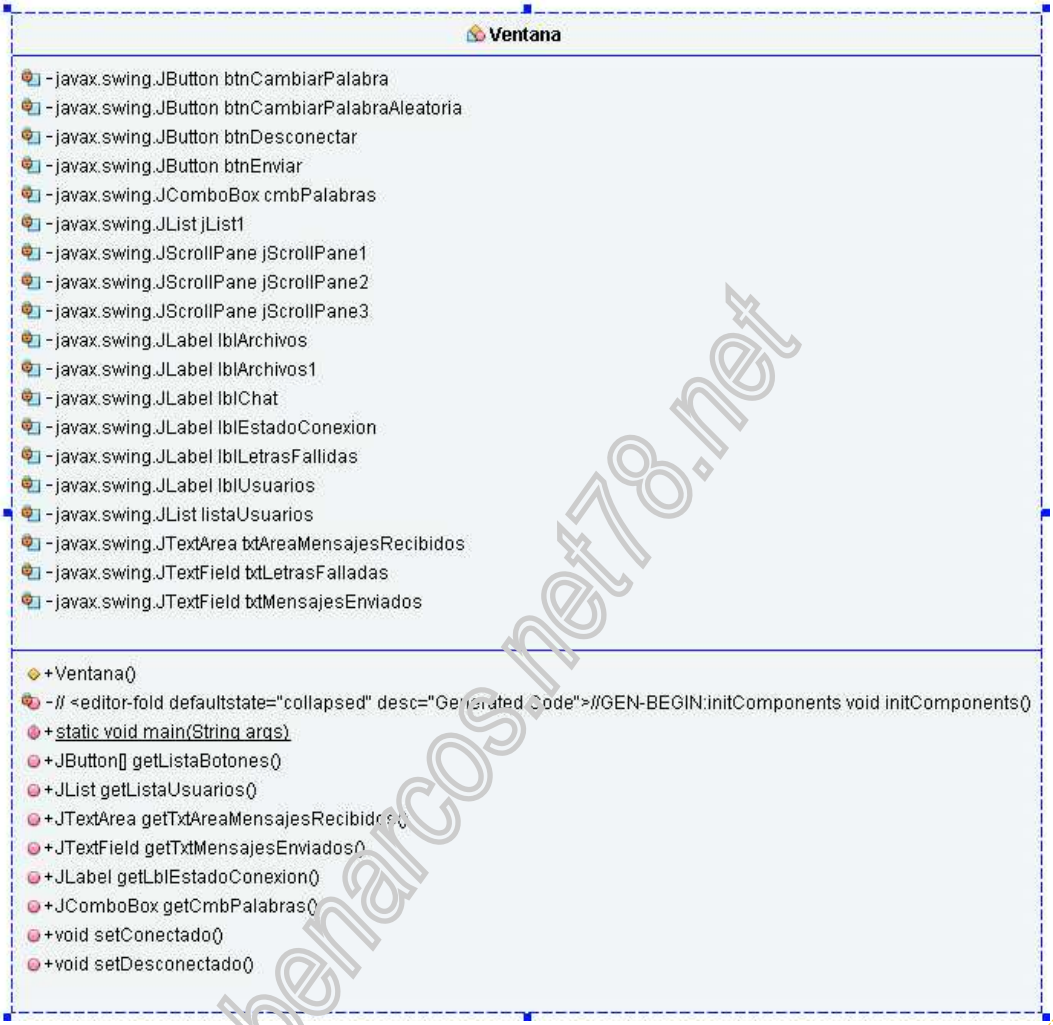
Serrvidor – Lógica y control



Servidor comunicación – Hilo de comunicación por usuario



Servidor GUI – Interfaz gráfica



3.4 Diseño de la interfaz gráfica y flujo de uso

La interfaz del usuario se ha diseñado previamente a la construcción de la solución aportando el siguiente esquena:

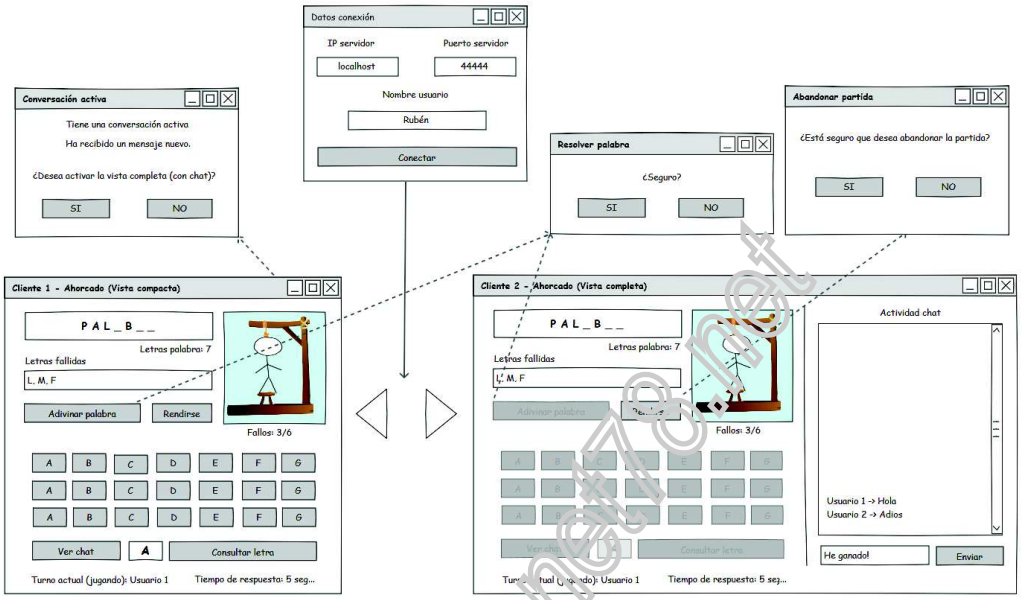


Ilustración 1 - Aplicación cliente, ventana de conexión y mensajes de aviso al usuario

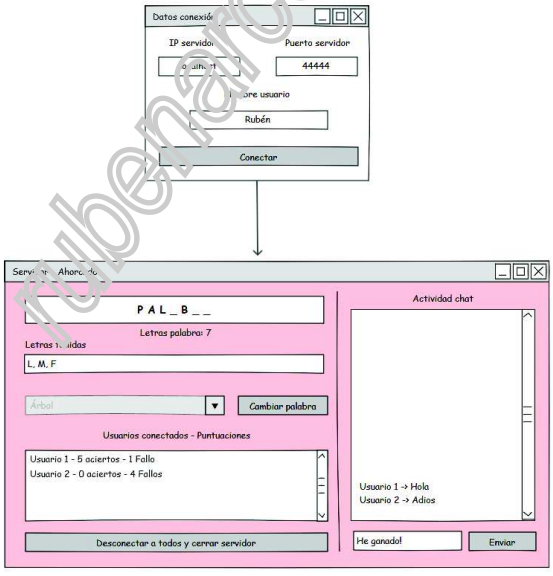


Ilustración 2 - Aplicación servidor y ventana de conexión

En ambos casos se ha sustituido finalmente las ventanas unificadas de conexión, por ventanas emergentes individuales por cada parámetro.

4 Programación fuente

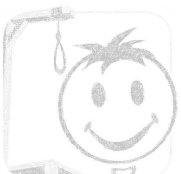
Se adjuntan, como solución de proyecto para NetBeans 8.0.2 compilación con Java HotSpot(TM) 64-Bit Server VM 25.11-b03.

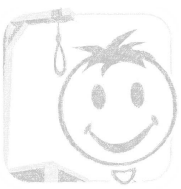
5 Justificación de la solución

Del análisis realizado en apartados anteriores acerca de la problemática que nos ocupa así como de los objetivos a lograr, se desprende que las necesidades de la gestión del juego del ahorcado mediante comunicación en red son muy específicas, por lo que la aplicación actual no satisface dichas necesidades en su totalidad.

Es necesario, por tanto, recurrir al diseño personalizado de la aplicación, utilizando para ello un entorno de desarrollo adecuado que facilite la creación de la misma. En este caso particular se ha reducido el número de condicionantes a tener en cuenta para la realización de las asignaciones de estados y consultas de los usuarios para poder gestionarse de la forma más veraz posible mediante la programación con técnicas de comunicación mediante protocolo TCP/IP con socket en Java. La utilización del protocolo propio mediante tipos enumerados, tanto en el servidor como en el cliente, se ha considerado la más adecuada por su sencillez y rapidez de implementación, aunque también se ha barajado la posibilidad de realizar un control mediante una capa adicional de comunicación basada en el envío de información serializada exclusivamente, lo cual simplificaría el código y su comprensión por tanto. La utilización de un objeto que contenga los datos de la partida se ha contemplado como un objeto independiente para mantener la coherencia en el nivel de abstracción seleccionado y la compatibilidad entre cliente y servidor, teniendo que ser exactamente la misma clase, para que el identificador UUID coincida tanto en el envío como en la recepción, no ha sido necesario las implementación explícita de una constante UUID en ambas clases, puesto que se ha contemplado desde el inicio del diseño para evitar esta especificación. Este es el mismo hecho por el cual se han separado los elementos visuales (ventanas de información) para mantener el patrón de diseño lo más cercano al modelo-vista-controlador.

Otro aspecto a tener en cuenta es la estructura en la que se encuentra el proyecto, la cual está pensada en la posterior escalabilidad del mismo, la reutilización de objetos y la incorporación de nuevas funcionalidades. Intentando por este hecho disponer de la mayor parte de elementos dinámicos, como el uso de constantes e instanciación de objetos desde el punto de inicio de la aplicación *(Cliente / Servidor)* en proyectos independientes.





6 Bibliografía

- Documentación propia y facilitada en clase.