Actividad

AE2

Ubicación

Tema 4

Objetivos

- Comprender el concepto de socket.
- Aprender a crear programas servidor y cliente.
- Saber crear conexiones y comunicaciones entre cliente y servidor.
- Aprender a gestionar servidores multihilo.

Temporalización

La duración prevista para esta actividad es de seis sesiones lectivas.

Instrucciones

Se solicita crear una aplicación de tipo chat cliente-servidor multicanal.

Inicialmente, el cliente se conectará al servidor, y este le enviará los canales disponibles.

Los canales disponibles deben leerse de un archivo de texto llamado “canales.txt”, que contendrá los nombres de los canales, uno por línea. Los canales disponibles son los siguientes: 1-cine, 2-música, 3-videojuegos y 4-literatura. El número de canales es fijo y no se pueden eliminar ni añadir canales al archivo.

El cliente debe elegir un canal de la lista y enviar su selección al servidor.

A continuación, el cliente enviará al servidor el nombre de usuario con el que desea participar en el canal (el nombre no puede contener espacios).

El servidor debe mantener una lista con los participantes de cada canal. Comprobará si el nombre proporcionado por el cliente ya existe. Si existe, el cliente deberá especificar otro nombre. Si no existe, lo añadirá a la lista del canal y devolverá el mensaje “OK”.

Una vez validado en el canal, el cliente podrá enviar mensajes al servidor. Estos mensajes pueden ser de 5 tipos:

- **String “whois”:** si el servidor recibe este string, devolverá al cliente un string que contenga los nombres de todos los clientes que están actualmente conectados al canal.
- **String “blablabla”:** el contenido del mensaje (blablabla) se envía a todos los usuarios conectados al canal en el que está el usuario.
- **String “@canalX blablabla”:** en este caso, el contenido del mensaje (blablabla) se enviará a los usuarios del canalX.
- **String “exit”:** el cliente abandona el chat y cierra la conexión con el servidor.

- **String "channels":** si el servidor recibe este string, devolverá al cliente un string que contenga los nombres de todos los canales disponibles.

En paralelo, el cliente podrá recibir mensajes de otros usuarios (de su canal o de otros) en cualquier momento (siempre a través del servidor). Por lo tanto, debes tener en cuenta que el cliente deberá tener dos hilos de ejecución: uno para enviar mensajes en cualquier momento y otro para recibir mensajes en cualquier momento.

Todos los mensajes (enviados y recibidos) se mostrarán en la misma consola. Cada mensaje que se envíe o reciba debe ir acompañado de una marca de tiempo (timestamp) cuando aparezca en la consola (el formato del timestamp es libre).

El servidor será multihilo y abrirá un hilo por cada cliente que se conecte. Para mantener a los clientes correctamente identificados en cada canal, deberás crear una lista (ArrayList) con los hilos que estén conectados por cada canal, lo que servirá para responder a las peticiones del cliente.

Cuando un cliente se conecte, se añadirá a la lista del canal, y cuando se desconecte, se eliminará de la lista. Además, cada vez que crees un nuevo hilo-servidor para gestionar un cliente, a ese hilo deberás pasarle la lista con todos los hilos del canal para que pueda saber quiénes son y enviarles mensajes.

Cada vez que el hilo-servidor reciba un mensaje del cliente, deberá identificar qué tipo de mensaje es para generar la respuesta adecuada y enviarla a quien corresponda.

Sugerencias (no son requisitos de la actividad):

Para mostrar mensajes propios o mensajes de otros clientes, puedes alternar entre System.out.println y System.err.println para que cambie el color de las letras en la consola.

El cliente usará la misma consola para mostrar los mensajes que envía y los que recibe. Si has realizado algunas pruebas de concurrencia, habrás comprobado que esto puede resultar incómodo si se reciben mensajes mientras se está escribiendo (se mezcla todo en la consola).

Una forma más ordenada de hacerlo es que los mensajes que el cliente envía se escriban desde una ventana emergente (popup). Por ejemplo, cada vez que el cliente desee enviar un mensaje, puede presionar la tecla Intro para que aparezca la ventana emergente.

Estructura sugerida para el proyecto (puedes plantearla de otras formas):

- Aplicación **cliente**:
  - Clase principal (con método main):

    Necesitará un BufferedReader y un PrintWriter para la parte de conexión al canal, y un PrintWriter (puedes utilizar el mismo) para enviar los mensajes al servidor.

  - Clase auxiliar (hilo, sin método main):

    Necesitará un BufferedReader para recibir y mostrar en la consola del cliente los mensajes del servidor (incluyendo los mensajes que envíen los otros clientes).

- Aplicación **servidor**:
  - Clase principal (con método main):

Crea el ServerSocket y va admitiendo clientes a través de hilos, gestionándolos con una lista, que pasará a cada hilo que cree.

- Clase auxiliar (hilo, uno por cada cliente, sin método main):

  Necesitará un BufferedReader y un PrintWriter para enviar/recibir mensajes desde el cliente al que está conectado y hacia cada uno de los clientes que tendrá en las listas de los diferentes canales. Controlará qué enviar y a quién según el mensaje que reciba del cliente.

Evaluación

La actividad es obligatoria y puede realizarse de forma individual o en parejas. Si se realiza en parejas, ambos alumnos deben realizar la entrega e indicarlo claramente como un comentario en la entrega.

Para la evaluación, se tendrá en cuenta el funcionamiento del programa, la codificación adecuada y la documentación del mismo. También se solicitará a los alumnos que presenten y expliquen en clase la aplicación que hayan desarrollado.

El código y la documentación deberán entregarse en Florida Oberta (archivo .zip) y también estar disponibles en el GitHub del alumno. Asimismo, se deberá entregar un vídeo corto que muestre la ejecución de las funcionalidades desarrolladas.

IMPORTANTE: si no se entrega el vídeo demostrativo, la actividad no será corregida.

Recursos

Material del módulo (Florida Oberta).

Rúbrica

Conexión a un canal	(0) No implementada.	(0.5) Falta controlar algún aspecto (por ejemplo, que el cliente ya esté conectado).		(1) La conexión funciona correctamente.
Comunicación cliente-servidor	(0) No implementada.	(1) Falta implementar bastantes mensajes.	(2) Falta implementar algún mensaje.	(3) Se implementan correctamente todos los mensajes.
Comunicación servidor-cliente	(0) No implementada.	(1) Envía mensajes solo al cliente que se ha conectado y faltan implementar funcionalidades según el tipo de mensaje del cliente.	(2) Envía mensajes solo al cliente que se ha conectado o faltan implementar funcionalidades según el tipo de mensaje del cliente.	(3) Envía mensajes a todos los clientes conectados según las indicaciones del mensaje. El cliente recibe y muestra correctamente todos los mensajes.
Gestión de las listas de hilos	(0) No implementada.	(0,5) Se implementa parcialmente o presenta errores.		(1) Las listas de hilos se gestionan correctamente.
Timestamp	(0) No implementada.	(0,5) Falta implementar los timestamps para los mensajes de salida o entrada.		(1) Se implementa correctamente.
Documentación	(0) No generada.	(0,25) Falten métodos por documentar y/o la descripción de clases y métodos no es adecuada.		(0,5) La documentación es correcta.
Calidad del código	(0) No sigue la guía de estilo.	(0,25) El código presenta errores de estilo.		(0,5) El código responde a la guía de estilo.