Documentación Chat Multi-Usuario

Chat multi-usuario empleando sockets y threads en Java

Eder Martínez Castro

Desarrollo de Aplicaciones Multiplataforma

21 de enero de 2026

Tabla de contenidos

1 Objetivo

El objetivo de esta práctica es **desarrollar** un **chat multi-usuario** en `Java` y `JavaFX` para la interfaz, utilizando **sockets** para la comunicación cliente-servidor. Donde se busca que múltiples usuarios puedan conectarse simultáneamente y enviar mensajes que lleguen al resto de usuarios en tiempo real.

2 Análisis y Especificación de Requisitos

2.1 Descripción del problema

Se necesita desarrollar una aplicación de chat que tenga lo siguiente:

- Un **servidor**, que escuche conexiones entrantes en un puerto TCP.
- Que varios **usuarios** puedan conectarse con un nombre de usuario.
- Los mensajes enviados por un usuario, se envían a todos los usuarios conectados.
- La interfaz gráfica debe mostrar los mensajes diferenciando entre mensajes del **sistema**, **propios** y de **otros usuarios**.

2.2 Requisitos funcionales

A continuación, se listan los requisitos funcionales del sistema:

ID	Requisito
RF-01	El cliente debe permitir introducir un nombre y poder conectarse al servidor.
RF-02	El servidor debe aceptar múltiples conexiones simultáneas.
RF-03	El cliente debe enviar mensajes de texto al servidor.
RF-04	El servidor debe reenviar los mensajes recibidos a todos los usuarios conectados.
RF-05	El cliente debe mostrar mensajes del sistema cuando alguien se conecta o desconecta.
RF-06	El cliente debe permitir desconectarse escribiendo `salir`.
RF-07	La interfaz debe diferenciar visualmente los mensajes propios, de los de los demás usuarios y de información.

Tabla 1. Requisitos funcionales del chat multi-usuario

2.3 Requisitos no funcionales

A continuación, se listan los requisitos no funcionales del sistema:

ID	Requisito
RNF-01	El sistema debe soportar un máximo de 10 usuarios concurrentes.
RNF-02	El servidor debe manejar desconexiones de forma segura liberando los recursos.

Tabla 2. Requisitos no funcionales del chat multi-usuario

3 Arquitectura del Sistema

3.1 Arquitectura General

La aplicación emplea la arquitectura **cliente-servidor** utilizando **sockets**.

- **Servidor (Server)**:
  - Abre un `ServerSocket` en el puerto `8080` y acepta conexiones.
  - Atiende a cada usuario de forma concurrente (hilos) para permitir varios usuarios al mismo tiempo.
  - Reenvía los mensajes a todos los usuarios conectados mediante `broadcast`.
- **Cliente (JavaFX)**:
  - Se organiza con **MVC** (Modelo: `ChatMessage`, Vista: `FXML/CSS`, Controlador: `ChatController`).
  - Se conecta al servidor mediante un `Socket`.
  - La obtención de los mensajes se maneja en otro **hilo** para no bloquear la UI.
- **Comunicación**:
  - Los mensajes se envían como **líneas de texto**.
  - Al conectarse, el usuario envía primero su **nombre**.
  - El comando `salir` desconecta al usuario.

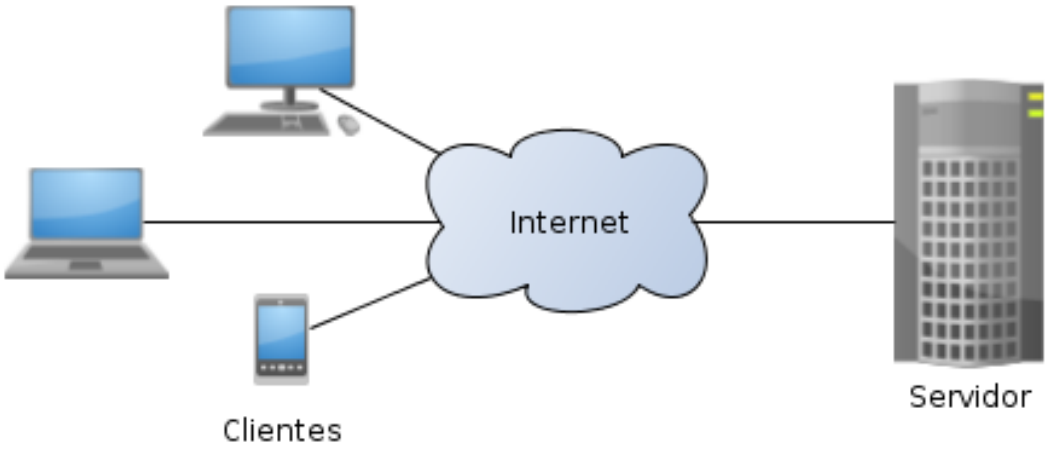

Figura 1. Diagrama de la arquitectura cliente-servidor

3.2 Clases

A continuación se describen las clases utilizadas en el chat multi-usuario:

3.2.1. Server

La clase `Server` implementa el servidor del chat, su función principal es la de **escuchar conexiones** y gestionar varios usuarios de forma concurrente:

- Inicia un `ServerSocket` en el puerto `8080` y permanece en un bucle aceptando conexiones con `accept()`.
- Por cada usuario conectado, obtiene un `Socket` y lo asigna a una tarea del **pool de hilos**.
- Guarda el `PrintWriter` de cada usuario conectado para poder enviar el mismo mensaje a todos.

Concurrencia: se utiliza un `FixedThreadPool` con un máximo de `10` hilos, atendiendo varios usuarios a la vez sin crear hilos ilimitados.
Difusión de mensajes: el método `broadcast(...)` recorre `writers` y envía el mensaje a todos los usuarios que estén conectados.

3.2.2. UserManager

`UserManager` es una clase interna que se encarga de gestionar a un usuario conectado:

- Obtiene el nombre del usuario y añade su `PrintWriter` a la lista `writers` para poder enviarle mensajes aparte de notificar la conexión del usuario.
- Escucha los mensajes en un bucle con `readLine()`.
- Si recibe `salir`, desconecta al usuario y notifica la desconexión.
- Al terminar, se elimina el `PrintWriter` del usuario de la lista `writers`, se notifica la desconexión y se cierra el socket.

3.2.3. Client

La clase `Client` se encarga de establecer y mantener la conexión con el servidor mediante un `Socket`, además de enviar y recibir mensajes:

- En `connect(...)` se crea el socket y se inicializan los streams de entrada/salida usando UTF-8.
- Envía el nombre del usuario al servidor.
- Permite enviar mensajes con `send(...)` y cerrar la conexión con `close()`.

Uso de Thread listener: para no bloquear la interfaz, al conectar se inicia un hilo que escucha mensajes del servidor con `readLine()` y los entrega al controlador usando `Consumer<String>message`.

3.2.4. ChatController

`ChatController` es el controlador JavaFX encargado de gestionar la interfaz y coordinar la comunicación con el servidor:

- Valida el nombre y llama a `client.connect(...)`.
- Habilita o deshabilita elementos de la interfaz según si está conectado o no.
- Envía mensajes con `client.send(...)` y limpia el campo para escribir el mensaje.
- Si el usuario escribe `salir`, envía el comando al servidor, cierra el socket y la aplicación.

Actualización segura de la interfaz: como los mensajes se reciben desde otro hilo, el controlador usa `Platform.runLater(...)` para añadir los mensajes en el `ListView` evitando así errores de concurrencia en JavaFX.

3.2.5. ChatMessage

`ChatMessage` es el modelo de datos que utiliza la interfaz para mostrar los mensajes. Incluye los diferentes tipos de mensaje (`INFO`, `ME`, `OTHER`) esto permite aplicar los estilos adecuados a la vista:

- `INFO`: mensajes del sistema (conexión/desconexión).
- `ME`: mensajes enviados por el propio usuario.
- `OTHER`: mensajes enviados por otros usuarios.

3.3 Stack empleado

Para el desarrollo de nuestra aplicación hemos utilizado las siguientes tecnologías:

Tecnología	Uso
Java	Lenguaje utilizado en el desarrollo de la aplicación.
JavaFX	Interfaz gráfica del cliente.
Sockets	Comunicación cliente-servidor orientada a conexión.
Threads / ExecutorService	Concurrencia para atender múltiples usuarios y poder escuchar los mensajes sin bloquear interfaz.
FXML + CSS	Definición de la interfaz y estilos.

Tabla 3. Tecnologías utilizadas

3.4 Interfaz

La interfaz gráfica está compuesta por los siguientes elementos:

- **Header**: formado por un campo para escribir el nombre y el botón de conexión.
- **Lista de mensajes**: `ListView` con celdas personalizadas para mostrar todos los distintos tipos de mensajes:
  - **Mensajes del sistema**: mensajes centrados con color destacado.
  - **Mensajes propios**: mensajes alineados a la derecha.
  - **Mensajes de otros usuarios**: mensajes alineados a la izquierda.
- **Footer**: formado por un campo para escribir mensajes y el botón de envío.

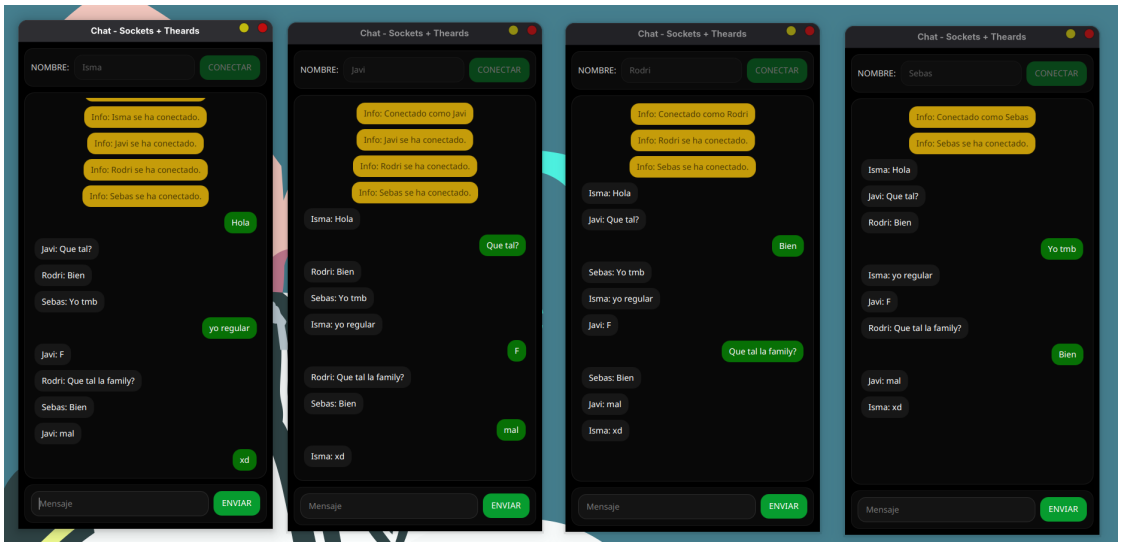

Figura 2. Interfaz gráfica del chat multi-usuario