# Implementación de un Chat en Tiempo Real en Java

Vamos a desarrollar un **chat en tiempo real** donde varios clientes pueden conectarse y enviarse mensajes entre sí a través de un servidor central. Implementaremos dos versiones:

1. **Chat basado en TCP Multicliente** *(Usaremos hilos para gestionar cada conexión).*
2. **Chat con WebSockets** *(Más eficiente y reactivo, ideal para aplicaciones web y móviles).*

---

## 1.- Chat TCP Multicliente (Usando Threads)

**Concepto**:

- El servidor manejará múltiples clientes usando hilos.
- Cada cliente puede enviar mensajes y recibir respuestas de otros clientes conectados.

---

### Código del Servidor TCP

**Pasos del Servidor**:

1. Escucha conexiones en un puerto específico.
2. Cada vez que un cliente se conecta, crea un nuevo hilo para gestionarlo.
3. Mantiene una lista de clientes activos para retransmitir mensajes a todos.

**Código: `ServidorChatTCP.java`**

```
package com.miapp.chat;


import java.io.*;

import java.net.ServerSocket;

import java.net.Socket;
```

```
import java.util.HashSet;

import java.util.Set;


public class ServidorChatTCP {

    private static final int PUERTO = 5000;

    private static Set<PrintWriter> clientes = new HashSet<>();


    public static void main(String[] args) {

        System.out.println("Servidor de Chat iniciado...");

        try (ServerSocket serverSocket = new ServerSocket(PUERTO)) {

            while (true) {

                Socket cliente = serverSocket.accept();

                System.out.println("Nuevo cliente conectado: " +
cliente.getInetAddress());


                // Crear y ejecutar un nuevo hilo para el cliente

                new Thread(new ManejadorCliente(cliente)).start();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }


    private static class ManejadorCliente implements Runnable {

        private Socket socket;

        private PrintWriter salida;
```

```
        private String nombreCliente;


        public ManejadorCliente(Socket socket) {

            this.socket = socket;

        }


        @Override

        public void run() {

            try (BufferedReader entrada = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

                 PrintWriter salida = new
PrintWriter(socket.getOutputStream(), true)) {


                this.salida = salida;

                clientes.add(salida);


                salida.println("Bienvenido al chat, introduce tu nombre:");

                nombreCliente = entrada.readLine();

                System.out.println(nombreCliente + " se ha unido al
chat.");

                enviarMensajeGlobal("📢 " + nombreCliente + " se ha unido
al chat.");


                String mensaje;

                while ((mensaje = entrada.readLine()) != null) {

                    System.out.println(nombreCliente + ": " + mensaje);
```

```
                    enviarMensajeGlobal("💬 " + nombreCliente + ": " +
mensaje);
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                if (salida != null) {
                    clientes.remove(salida);
                }
                enviarMensajeGlobal("X" + nombreCliente + " ha salido del
chat.");
                System.out.println(nombreCliente + " desconectado.");
                try {
                    socket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        private void enviarMensajeGlobal(String mensaje) {
            for (PrintWriter cliente : clientes) {
                cliente.println(mensaje);
            }
        }
    }
```

```
}
```

---

## Código del Cliente TCP

## Pasos del Cliente:

1. Se conecta al servidor y proporciona un nombre de usuario.
2. Envía y recibe mensajes en tiempo real.
3. Puede escribir mensajes en la consola y recibir mensajes de otros clientes.

**Código: `ClienteChatTCP.java`**

```
package com.miapp.chat;



import java.io.*;

import java.net.Socket;



public class ClienteChatTCP {

    public static void main(String[] args) {

        try (Socket socket = new Socket("localhost", 5000);

             BufferedReader entrada = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

             PrintWriter salida = new PrintWriter(socket.getOutputStream(),
true);

             BufferedReader teclado = new BufferedReader(new
InputStreamReader(System.in))) {



            System.out.println("Conectado al chat. Introduce tu nombre:");

            String nombre = teclado.readLine();

            salida.println(nombre);
```

```
            // Hilo para recibir mensajes del servidor
            new Thread(() -> {
                try {
                    String mensaje;
                    while ((mensaje = entrada.readLine()) != null) {
                        System.out.println(mensaje);
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }).start();

            // Enviar mensajes al chat
            String mensaje;
            while ((mensaje = teclado.readLine()) != null) {
                salida.println(mensaje);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

**Ahora, puedes ejecutar el servidor y abrir múltiples clientes para probar el chat en tiempo real.**

# 2.- Chat en Tiempo Real con WebSockets

WebSockets son **más eficientes** que TCP tradicional porque:

- Permiten una **conexión persistente** entre cliente y servidor.
- Son **más rápidos** porque no requieren establecer una conexión nueva para cada mensaje.
- Son **ideales para aplicaciones web y móviles**.

---

## Código del Servidor WebSockets (Spring Boot)

### 1.- Agregar Dependencias en `pom.xml`

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
```

### 2.- Configurar WebSockets 📌 Creamos la configuración del servidor WebSocket.

**Código: `WebSocketConfig.java`**

```
package com.miapp.chat;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.config.annotation.*;

@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {
```

```
    @Override

    public void registerWebSocketHandlers(WebSocketHandlerRegistry
registry) {

        registry.addHandler(new ChatWebSocketHandler(),
"/chat").setAllowedOrigins("*");

    }

}
```

**3.- Implementar el Manejador de Mensajes - Manejador que retransmite los mensajes a todos los clientes.**

**Código: `ChatWebSocketHandler.java`**

```
package com.miapp.chat;


import org.springframework.web.socket.*;

import org.springframework.web.socket.handler.TextWebSocketHandler;


import java.util.HashSet;

import java.util.Set;


public class ChatWebSocketHandler extends TextWebSocketHandler {

    private static final Set<WebSocketSession> sesiones = new HashSet<>();


    @Override

    public void afterConnectionEstablished(WebSocketSession session) throws
Exception {

        sesiones.add(session);
```

```
        session.sendMessage(new TextMessage("Bienvenido al chat en tiempo
real!"));

    }



    @Override

    protected void handleTextMessage(WebSocketSession session, TextMessage
message) throws Exception {

        for (WebSocketSession s : sesiones) {

            s.sendMessage(new TextMessage(message.getPayload()));

        }

    }



    @Override

    public void afterConnectionClosed(WebSocketSession session, CloseStatus
status) throws Exception {

        sesiones.remove(session);

    }

}
```

## Código del Cliente WebSocket

**Cliente WebSocket en Java con Spring WebFlux.**

```
package com.miapp.cliente;



import
org.springframework.web.reactive.socket.client.ReactorNettyWebSocketClient;
```

```
import org.springframework.web.reactive.socket.WebSocketMessage;

import reactor.core.publisher.Mono;


import java.net.URI;

import java.util.Scanner;


public class ClienteChatWebSocket {

    public static void main(String[] args) {

        ReactorNettyWebSocketClient client = new
ReactorNettyWebSocketClient();

        Scanner scanner = new Scanner(System.in);


        client.execute(URI.create("ws://localhost:8080/chat"), session ->

            session.send(Mono.just(session.textMessage("¡Hola, soy un nuevo
usuario!")))

                .thenMany(session.receive()

                    .map(WebSocketMessage::getPayloadAsText)

                    .doOnNext(System.out::println))

                .then()

        ).subscribe();


        while (true) {

            String mensaje = scanner.nextLine();

            client.execute(URI.create("ws://localhost:8080/chat"), session
->


session.send(Mono.just(session.textMessage(mensaje))).then()
```

```
            ).subscribe();

        }

    }

}
```

**Ahora, puedes ejecutar el servidor y conectar varios clientes al chat en tiempo real.**