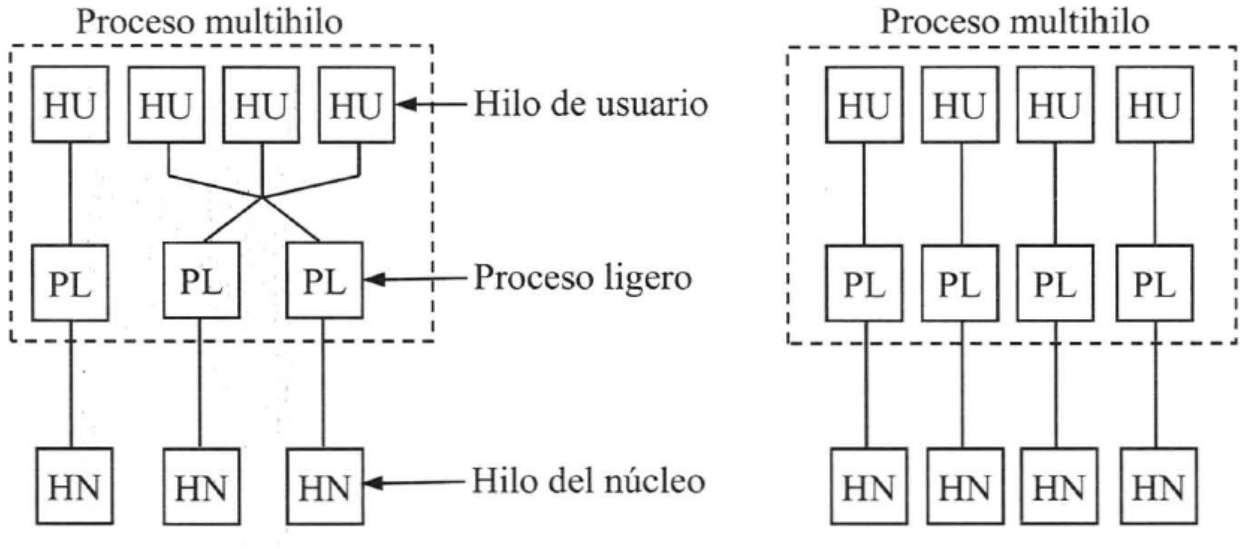
Sincronización de hilos

La sincronización multihilo es un concepto clave en la programación concurrente, que se refiere al control del acceso concurrente a recursos compartidos por múltiples hilos de ejecución. Cuando varios hilos acceden a un mismo recurso (como una variable o un objeto), es fundamental controlar este acceso para evitar condiciones de carrera, inconsistencias y comportamiento impredecible.

La sincronización garantiza que un solo hilo tenga acceso a un recurso crítico en un momento dado, bloqueando temporalmente a otros hilos hasta que el recurso quede disponible nuevamente.



¿Para qué sirve la sincronización multihilo?

- **Evitar condiciones de carrera:** Las condiciones de carrera ocurren cuando dos o más hilos intentan acceder o modificar un recurso compartido simultáneamente, lo que puede llevar a resultados inesperados o incorrectos.
- **Proteger recursos compartidos:** En aplicaciones multihilo, a menudo hay recursos (como archivos, variables, conexiones, etc.) que deben ser manipulados por un solo hilo a la vez. La sincronización asegura que estos recursos sean utilizados de manera segura.
- **Coordinación entre hilos:** En algunos casos, los hilos deben cooperar y esperar unos a otros para completar una tarea. La sincronización permite coordinar la ejecución de estos hilos para asegurar que las operaciones ocurran en el orden correcto.

¿De qué se compone la sincronización multihilo?

- **Secciones críticas:** Son las partes del código que acceden a recursos compartidos que deben ser protegidos para que solo un hilo pueda ejecutarlas a la vez.
- **Locks (bloqueos):** Son mecanismos que permiten que un hilo "bloquee" una sección crítica mientras la usa, para evitar que otros hilos la modifiquen al mismo tiempo. En Java, se pueden implementar usando bloques synchronized o clases especializadas como Lock.

- **Monitores:** Un monitor es un mecanismo que permite que solo un hilo a la vez ejecute una sección crítica de código. En Java, cada objeto tiene un monitor, y los bloques synchronized están basados en este mecanismo.
- **Condiciones de espera y notificación:** En situaciones donde un hilo debe esperar a que otro termine su tarea antes de continuar, Java proporciona los métodos wait(), notify(), y notifyAll() para la coordinación entre hilos.

Desarrollo:

Para la presente práctica se va a desarrollar un chat que permite múltiples conexiones y que reciba los mensajes de una forma sincronizada en lenguaje Java.

Para ello comenzaremos creando una clase Chatservidor.java donde la lógica es como se muestra a continuación:

```
public static void main(String[] args) {
    System.out.println("Servidor de chat iniciado...");

    try (ServerSocket serverSocket = new ServerSocket(PORT)) {
        while (true) {
            new ClientHandler(serverSocket.accept()).start();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

El servidor inicia en el puerto 12345 (definido en la constante PORT). La línea new ServerSocket(PORT) crea un socket de servidor que escucha en ese puerto. La llamada a serverSocket.accept() bloquea la ejecución hasta que un cliente se conecte. Cuando un cliente se conecta, se crea una nueva instancia de ClientHandler (una clase interna que maneja la conexión con ese cliente) y se inicia en un nuevo hilo con start(), permitiendo que múltiples clientes se conecten simultáneamente sin que el servidor se detenga.

```
private static class ClientHandler extends Thread {
    private Socket socket;
    private PrintWriter out;
    private BufferedReader in;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }
}
```

Permite que cada cliente sea manejado en un hilo independiente. El constructor recibe un objeto Socket que representa la conexión con el cliente y lo almacena en el campo socket de la clase. Este socket se utilizará para establecer los flujos de entrada y salida para la comunicación con el cliente.

```
public void run() {
    try {
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(), true);

        synchronized (clientWriters) {
            clientWriters.add(out);
        }
```

El método run() es ejecutado cuando el hilo comienza. Dentro de este método, se configuran los flujos de entrada y salida del cliente:

- in es un BufferedReader que lee mensajes enviados por el cliente desde el socket de entrada.
- out es un PrintWriter que permite enviar mensajes al cliente.

La instrucción synchronized (clientWriters) garantiza que el acceso al conjunto clientWriters (que almacena a todos los clientes conectados) sea sincronizado, evitando conflictos si varios hilos acceden a este recurso al mismo tiempo.

```
String message;
while ((message = in.readLine()) != null) {
    System.out.println("Mensaje recibido: " + message);
    synchronized (clientWriters) {
        for (PrintWriter writer : clientWriters) {
            writer.println(message);
        }
    }
}
```

El servidor recibe los mensajes del cliente mediante el flujo de entrada in.readLine(), que lee una línea completa de texto. Mientras los mensajes lleguen, se imprime en la consola del servidor con System.out.println() para monitorear la actividad del chat. Luego, dentro de un bloque sincronizado, se itera sobre el conjunto de clientWriters y se retransmite el mensaje recibido a todos los clientes conectados.

Ahora, para el cliente se va a crear la clase Chatcliente.java que será el encargado de enviar los mensajes al servidor como se muestra a continuación:

```
private static final String SERVER_ADDRESS = "AQUI_VA_LA_IP_DEL_SERVIDOR";
private static final int SERVER_PORT = 12345;

public static void main(String[] args) {
    try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
         BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         Scanner scanner = new Scanner(System.in)) {
```

Se establece la conexión del cliente al servidor de chat. SERVER_ADDRESS y SERVER_PORT definen la dirección IP del servidor y el puerto en el que escucha el servidor.

Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT); crea un socket que se conecta al servidor utilizando la dirección y el puerto indicados.

```
// Hilo para recibir mensajes del servidor
Thread receiveMessages = new Thread(() -> {
    try {
        String message;
        while ((message = in.readLine()) != null) {
            System.out.println("Mensaje recibido: " + message);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
});
receiveMessages.start();
```

La variable message almacena las líneas de texto recibidas del servidor mediante in.readLine(), y si el servidor envía algún mensaje, este se muestra en la consola con System.out.println("Mensaje recibido: " + message).

El uso de un hilo permite que el cliente reciba mensajes del servidor de manera asíncrona mientras sigue interactuando con la interfaz del usuario (esperando que el usuario escriba nuevos mensajes).

```
// Enviar mensajes al servidor
System.out.println("Conectado al chat. Escribe tu mensaje:");
while (true) {
    String messageToSend = scanner.nextLine();
    out.println(messageToSend);
}
```

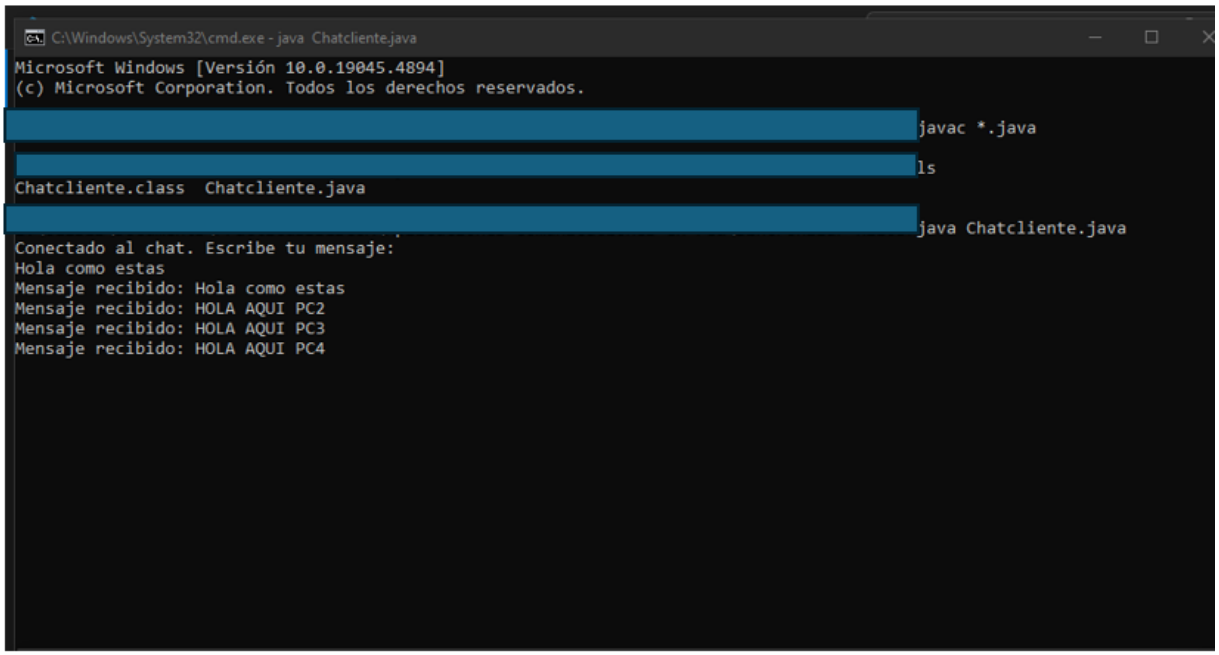
El programa imprime el mensaje "Conectado al chat. Escribe tu mensaje:", indicando que el cliente está listo para enviar mensajes.

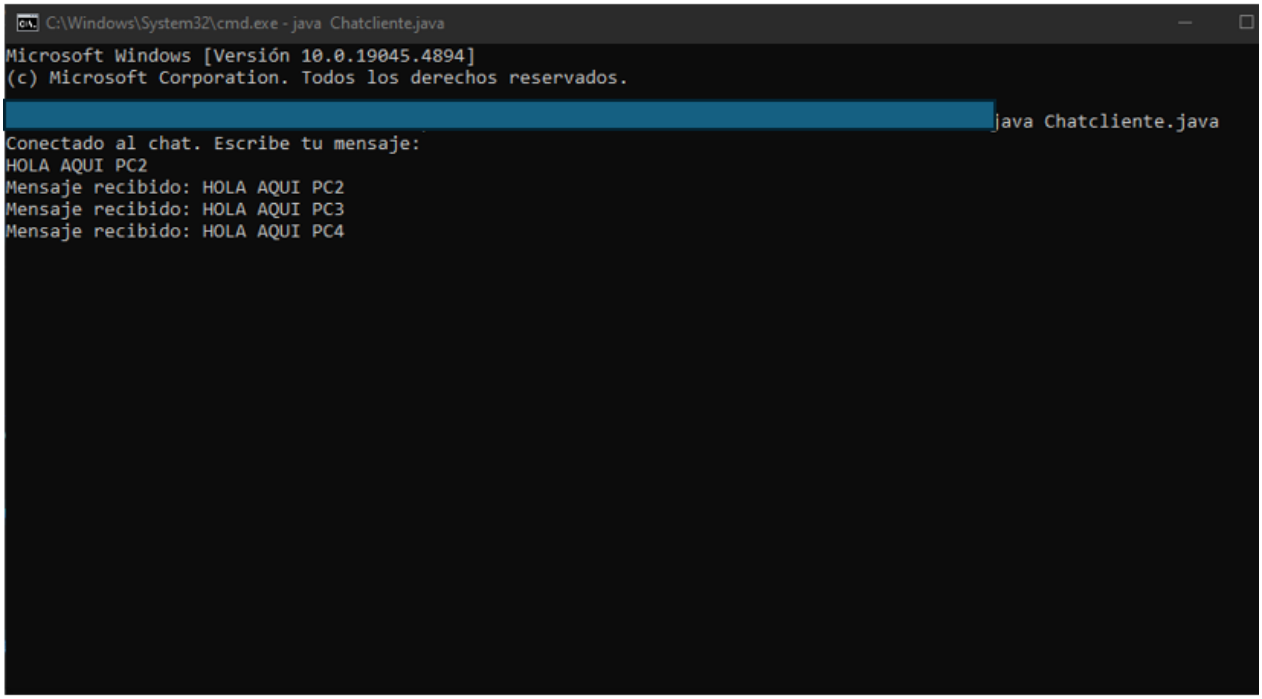
En un ciclo while (true), el cliente lee las entradas del usuario a través de scanner.nextLine() y envía cada mensaje capturado al servidor utilizando out.println(messageToSend). Este ciclo es infinito, lo que significa que el cliente seguirá funcionando hasta que se cierre manualmente.

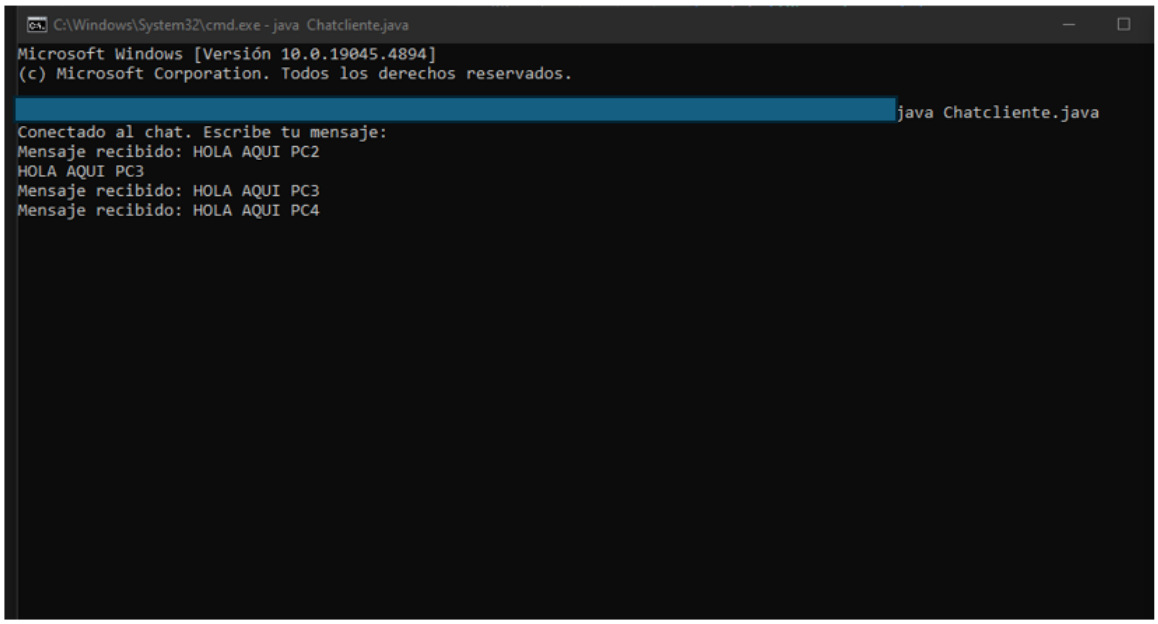
Una vez compilado con el siguiente comando:

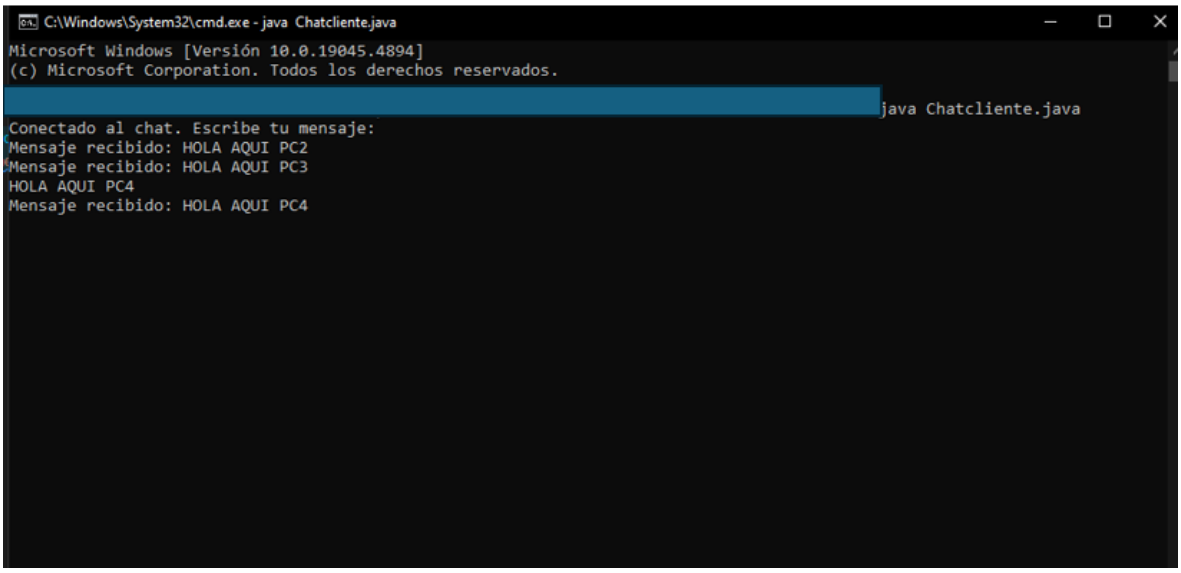
```
javac *.java
```

Ahora en la ejecución se tienen los siguientes cliente que como simulación podrían ser distintas personas o máquinas mandando mensaje al servidor:






C:\Windows\System32\cmd.exe - java Chatcliente.java
Microsoft Windows [Versión 10.0.19045.4894]
(c) Microsoft Corporation. Todos los derechos reservados.
java Chatcliente.java
Conectado al chat. Escribe tu mensaje:
Mensaje recibido: HOLA AQUI PC2
HOLA AQUI PC3
Mensaje recibido: HOLA AQUI PC3
Mensaje recibido: HOLA AQUI PC4


C:\Windows\System32\cmd.exe - java Chatcliente.java
Microsoft Windows [Versión 10.0.19045.4894]
(c) Microsoft Corporation. Todos los derechos reservados.
java Chatcliente.java
Conectado al chat. Escribe tu mensaje:
Mensaje recibido: HOLA AQUI PC2
Mensaje recibido: HOLA AQUI PC3
HOLA AQUI PC4
Mensaje recibido: HOLA AQUI PC4

El servidor podrá ver el mensaje de la siguiente forma: