**UNIVERSIDAD TÉCNICA DE MANABI**

**FACULTAD DE CIENCIAS INFORMÁTICAS**

**ESCUELA:**

Tecnología de la Información y Comunicación

**DOCENTE:**

Ing. Miguel Rodriguez, MSc.

**ESTUDIANTE:**

Rodríguez Delgado Alex Damián

**TEMA:**

**“Comunicación entre procesos distribuidos mediante uso de la red”**

ENLACE DEL VIDEO : <https://drive.google.com/drive/folders/1V8H-lcE7dvv-X7_ItbId_wkUosjDDlqm?usp=sharing>

Cliente:  
public class Cliente {  
 public static void main(String[] args) {  
 try {  
 int puerto = 5000;  
 Socket cliente = new Socket("localhost", puerto);  
 DataInputStream entrada = new DataInputStream(cliente.getInputStream());  
 DataOutputStream salida = new DataOutputStream(cliente.getOutputStream());  
 // Interacción con el usuario para elegir qué información desea  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println("¿Qué información desea obtener? (fecha/hora): ");  
 String opcion = scanner.nextLine().toLowerCase();  
 // Enviar la solicitud al servidor  
 salida.writeUTF(opcion);  
 // Recibir y mostrar la respuesta del servidor  
 String respuesta = entrada.readUTF();  
 System.*out*.println("Respuesta del servidor: " + respuesta);  
 cliente.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Se implementa un cliente que se conecta a un servidor en el puerto 5000 (en el mismo equipo local, ya que se especifica "localhost"). Después de establecer la conexión, el cliente solicita al usuario que elija qué tipo de información desea obtener: fecha o hora.

**Aquí está el flujo del programa:**

1. Se establece una conexión con el servidor en el puerto 5000.

2. Se crean flujos de entrada y salida para comunicarse con el servidor.

3. Se solicita al usuario que elija qué tipo de información desea obtener (fecha o hora).

4. La opción elegida por el usuario se envía al servidor.

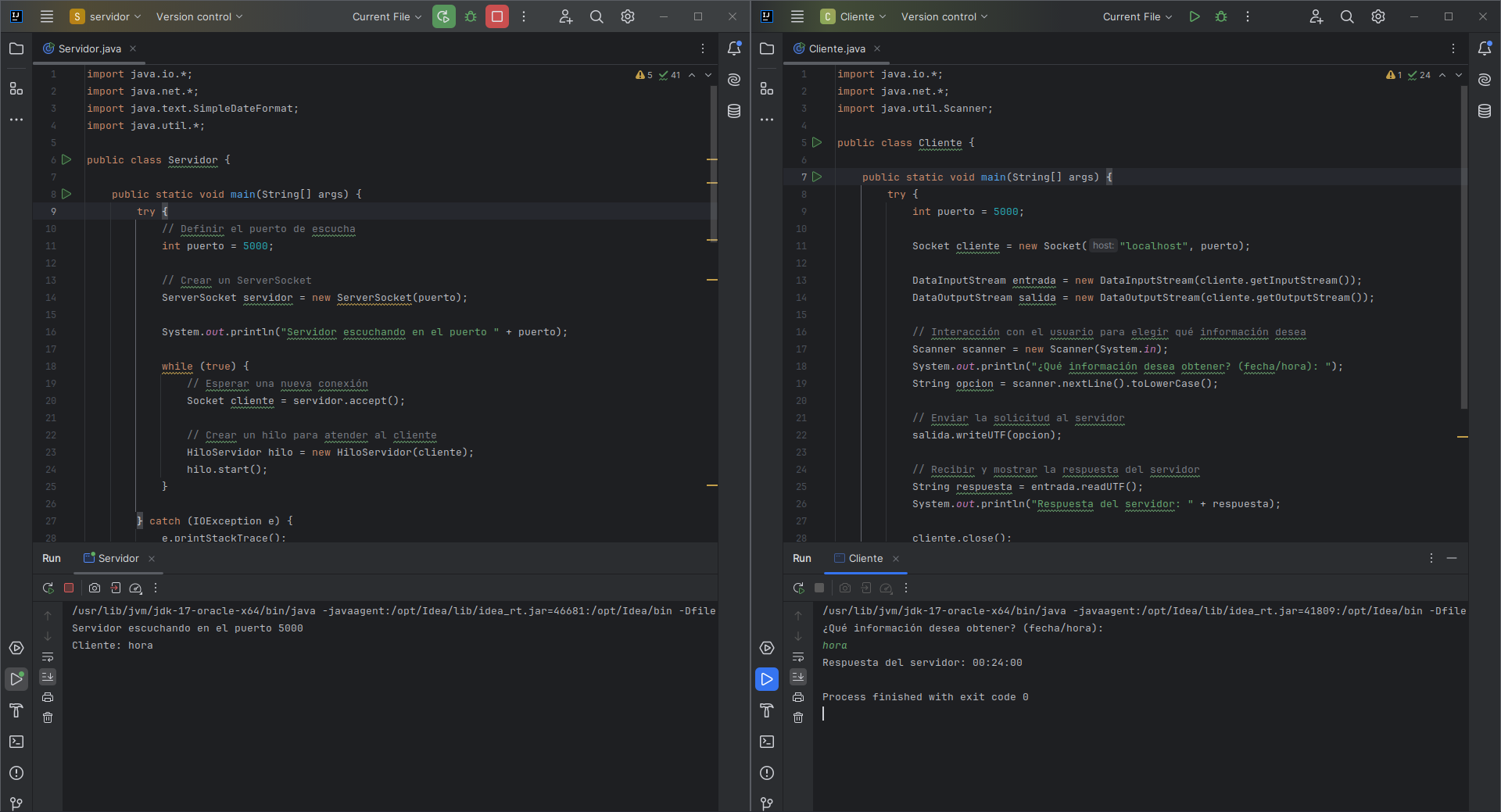
5. El cliente espera y recibe la respuesta del servidor.

6. La respuesta del servidor se muestra en la consola.

7. Finalmente, se cierra la conexión con el servidor.

El cliente utiliza un `Scanner` para obtener la entrada del usuario y envía esta entrada al servidor a través de un `DataOutputStream`. Luego, recibe la respuesta del servidor a través de un `DataInputStream` y la muestra en la consola. Si ocurre algún error durante la comunicación con el servidor, se imprime la traza de la excepción.

1



Servidor:

public class Servidor {  
 public static void main(String[] args) {  
 try {  
 // Definir el puerto de escucha  
 int puerto = 5000;  
 // Crear un ServerSocket  
 ServerSocket servidor = new ServerSocket(puerto);  
 System.*out*.println("Servidor escuchando en el puerto " + puerto);  
 while (true) {  
 // Esperar una nueva conexión  
 Socket cliente = servidor.accept();  
 // Crear un hilo para atender al cliente  
 HiloServidor hilo = new HiloServidor(cliente);  
 hilo.start();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}  
class HiloServidor extends Thread {  
 private Socket cliente;  
 public HiloServidor(Socket cliente) {  
 this.cliente = cliente;  
 }  
 @Override  
 public void run() {  
 try {  
 // Obtener los flujos de entrada y salida del socket  
 DataInputStream entrada = new DataInputStream(cliente.getInputStream());  
 DataOutputStream salida = new DataOutputStream(cliente.getOutputStream());  
 // Leer el mensaje del cliente  
 String mensaje = entrada.readUTF();  
 // Mostrar el mensaje recibido por consola  
 System.*out*.println("Cliente: " + mensaje);  
 // Obtener la fecha o la hora actual  
 String respuesta;  
 if (mensaje.equalsIgnoreCase("hora")) {  
 respuesta = new SimpleDateFormat("HH:mm:ss").format(new Date());  
 } else if (mensaje.equalsIgnoreCase("fecha")) {  
 respuesta = new SimpleDateFormat("dd/MM/yyyy").format(new Date());  
 } else {  
 respuesta = "Mensaje no válido";  
 }  
 // Enviar la respuesta al cliente  
 salida.writeUTF(respuesta);  
 // Cerrar el socket  
 cliente.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

2

Se implementa un servidor simple que escucha conexiones en el puerto 5000. Cuando un cliente se conecta, se crea un nuevo hilo para manejar esa conexión. El servidor espera mensajes del cliente y responde según el mensaje recibido.

**Aquí está el flujo del programa:**

El servidor se inicia en el puerto 5000.

Espera conexiones entrantes en un bucle infinito.

Cuando se recibe una conexión entrante, se crea un nuevo hilo (HiloServidor) para manejarla.

El hilo del servidor recién creado obtiene los flujos de entrada y salida del cliente.

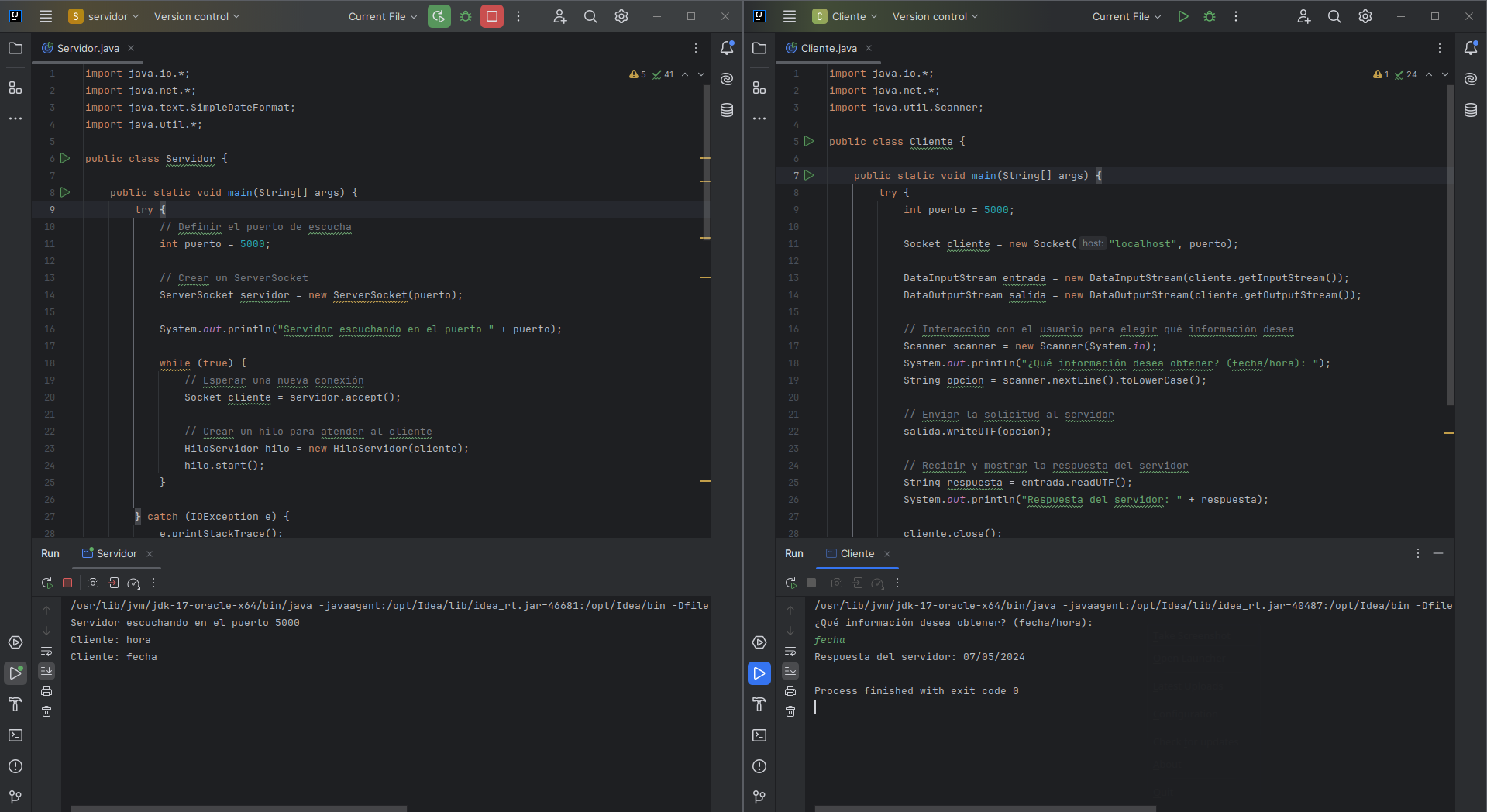
Lee el mensaje enviado por el cliente.

Basado en el mensaje recibido, el servidor determina si el cliente está solicitando la hora actual o la fecha.

Responde al cliente con la hora o la fecha actual según la solicitud.

Cierra la conexión con el cliente.

El código utiliza **DataInputStream** y **DataOutputStream** para leer y escribir datos desde y hacia el cliente respectivamente. También usa **SimpleDateFormat** para formatear la fecha y la hora actual antes de enviarlas de vuelta al cliente.



**Conclusiones:**

Ambos códigos ilustran la comunicación básica entre un cliente y un servidor utilizando sockets en Java. El servidor espera conexiones entrantes en un puerto específico y maneja cada conexión en un hilo separado, mientras que el cliente se conecta al servidor, envía una solicitud y espera la respuesta, esta arquitectura permite la comunicación bidireccional entre aplicaciones en diferentes máquinas o en la misma máquina.

El protocolo de comunicación entre el cliente y el servidor es simple pero efectivo, el cliente envía una cadena que indica si desea obtener la fecha o la hora, y el servidor responde con la información solicitada o un mensaje de error si la solicitud no es válida, este enfoque muestra cómo se pueden establecer protocolos simples para intercambiar datos entre aplicaciones cliente y servidor.