


## Taller de Programación de Aplicaciones Cliente-Servidor

Requisitos Previos:

- Conocimiento básico de programación en Java.
- Comprensión básica de redes y protocolos de comunicación.
- Ejemplos realizados en clases previas: [ejemplo0\\_tcp](#) 

### Actividades Propuestas:

#### PARTE 1: 30 minutos

1. Agregar Retardo al Servidor:

- Modificar el servidor para que, al recibir una solicitud del cliente, espere un tiempo antes de responder. Esto simulará un retardo en la respuesta del servidor.
- Los estudiantes deben decidir cómo implementar este retardo y configurar su duración.

2. Conexión Múltiple con Retardo:

- Luego de implementar el retardo en el servidor, los estudiantes deben probar la conexión de otro cliente mientras el servidor está ocupado atendiendo a un primer cliente.
- Esto permitirá comprobar cómo el servidor maneja múltiples conexiones y si el retardo afecta a los clientes adicionales.

3. Implementar Servidor Multihilo:

- Extender la clase Thread para crear una nueva clase que represente un hilo de ejecución del servidor.
- Modificar el servidor para que pueda manejar múltiples clientes de manera concurrente utilizando hilos.
- Cada hilo debe tener su propio retardo al responder a los clientes.

4. Prueba con Varios Clientes:

- Desarrollar un conjunto de clientes que puedan conectarse al servidor simultáneamente y enviar solicitudes.
- Realizar pruebas para verificar que el servidor pueda manejar correctamente múltiples conexiones concurrentes, cada una con su propio retardo.

Repetir el proceso para el ejemplo usando protocolo UDP: [ejemplo1\\_udp](#) 

#### PARTE 2:

Considere el aplicativo **ReproductorMusic.java** 

Este código implementa un reproductor de audio básico en Java utilizando la biblioteca `javax.sound.sampled`. Este código crea un reproductor de audio básico en Java que carga un archivo de audio desde el sistema de archivos y lo reproduce utilizando la biblioteca **`javax.sound.sampled`**. Aquí está la explicación del código proporcionado:

1. Se importan las clases necesarias:

```
java
import java.io.*;
import java.net.*;
import javax.sound.sampled.*;
```

2. Se define la clase **ReproductorMusic** que contiene la lógica principal del reproductor de audio.

```
public class ReproductorMusic
```

las variables de instancia **in** y **out** que representan el flujo de entrada de audio y la línea de datos de origen respectivamente. La variable **route** especifica la ruta del archivo de audio que se va a reproducir.

```
    AudioInputStream in; //datos de entrada
    SourceDataLine out; //salida a la tarjeta de audio
    private String route = "./songs/Song1_40k.wav";
```

4. El constructor **ReproductorMusic()** llama al método **initiateAudio()** para inicializar el flujo de audio y comenzar la reproducción.

```
    public ReproductorMusic() {
        initiateAudio();
    }
```

5. El método **initiateAudio()** inicializa el flujo de audio y la línea de datos de origen utilizando el archivo especificado en **route**. Se abre la línea de datos de origen y se inicia la reproducción del audio.

```
    public void initiateAudio() {
        try {
            File file = new File(route);
            in = AudioSystem.getAudioInputStream(file);

            out = AudioSystem.getSourceDataLine(in.getFormat());
            out.open(in.getFormat());
            out.start();
            playAudio();
        } catch (Exception e) {
            // TODO: handle exception
        }
    }
```

6. El método **playAudio()** lee los datos del archivo de audio y los escribe en la línea de datos de origen para reproducir el audio.

```
    private void playAudio() {
        byte[] buffer = new byte[1024];
        try {
            int count;
            while ((count = in.read(buffer, 0, buffer.length)) != -1) {
                if (count > 0) {
                    out.write(buffer, 0, count);
                }
            }
        }
```

```

    } catch (Exception e) {
        // TODO: handle exception
    }
}

```

Dedicar 5 minutos para entender la lógica del código anterior

7. El método `main()` crea una instancia del `ReproductorMusic` y llama al método `initiateAudio()` para iniciar la reproducción del audio.

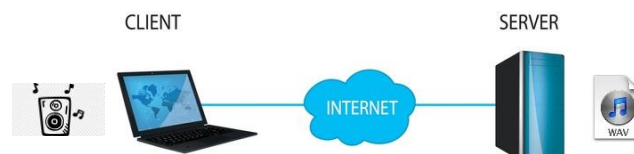
```

public static void main(String[] args) {
    new ReproductorMusic().initiateAudio();
}

```

### Actividad: 1 hora

1. compilar y ejecutar el código proporcionado. Modificar el código para que reciba la canción a reproducir por entrada del teclado.
2. Usar un reproductor externo para reproducir la canción marcada como \_16k, comparar como suena la canción marcada con \_44k y con el archivo mp3. Escucha alguna diferencia perceptible? Porque los archivos tienen diferente tamaño en disco?
3. Que información guarda un archivo de audio? Consultar la estructura de un archivo de audio tipo wav. Cual es la diferencia entre los diferentes tipos de archivo? Por ejemplo mp3 y wav.
4. Con el código modificado cargar y reproducir diferentes archivos de audio (mp3 y wav). Así podrán experimentar con la reproducción de diferentes tipos de archivos de audio y observar cómo afecta al funcionamiento del reproductor. Intentar reproducir un archivo que no existe. Como manejar de forma adecuada estas situaciones?
5. Investigar y aprender más sobre la biblioteca `javax.sound.sampled`, así como sobre otros aspectos relacionados con la manipulación de audio en Java. Consultar el funcionamiento de otras clases y herramientas con las cuales no esté familiarizado.
6. Pueden explorar cómo agregar funcionalidades adicionales al reproductor, como control de volumen, reproducción en bucle.
7. Discutir la estructura que debe tener un sistema cliente-servidor donde se tenga un host enviando audio y el otro host recibiendo y reproduciendo el audio. Que protocolo usaría TCP o UDP?



### Parte 3: 30 minutos

Usar el [\[enlace tablero miro\]](#) para proponer un diagrama con la estructura y funciones requeridas en el punto 7.