Guía de Planificación para Implementar una Aplicación Cliente-Servidor de Reproducción de Audio en Java utilizando DatagramSocket (UDP)

Objetivo:

Diseñar e implementar un sistema cliente-servidor que permita la transmisión y reproducción de archivos de audio en formato WAV desde el servidor al cliente utilizando sockets UDP en Java.

Requisitos Previos:

- Conocimientos básicos de programación en Java.

- Comprensión básica de redes y comunicación entre clientes y servidores.

- Familiaridad con el formato de archivo de audio WAV y sus características básicas.

1. Análisis de Requisitos:

- Identificar claramente los requisitos del sistema, incluyendo las funciones principales, como la transmisión de archivos de audio, la reproducción en el cliente, y los requisitos de comunicación entre cliente y servidor.

2. Diseño de la Arquitectura:

- Determinar la arquitectura general del sistema, incluyendo las clases necesarias tanto para el cliente como para el servidor.

- Decidir cómo se manejará la comunicación entre cliente y servidor utilizando sockets UDP.

- Considerar la necesidad de utilizar hilos para la recepción y reproducción de audio en el cliente.

3. Definición del Protocolo de Comunicación:

- Especificar un protocolo de comunicación entre cliente y servidor que permita la transmisión eficiente de archivos de audio en formato WAV.

- Determinar cómo se estructurarán los mensajes intercambiados entre cliente y servidor para la transmisión y reproducción del audio.

4. Identificación de las Características del Formato de Audio WAV:

- Investigar y comprender las características básicas del formato de archivo de audio WAV, incluyendo:

  - Frecuencia de muestreo: número de muestras de audio tomadas por segundo.

  - Número de bits por muestra: cantidad de bits utilizados para representar cada muestra de audio.

  - Número de canales: cantidad de canales de audio (por ejemplo, mono o estéreo).

  - Codificación: si los datos de audio están codificados como enteros con o sin signo y si se utilizan codificaciones big-endian o little-endian.

- Asegurarse de que el diseño del sistema tenga en cuenta estas características para garantizar la correcta transmisión y reproducción del audio.

5. Planificación de Implementación:

- Dividir el desarrollo del sistema en tareas más pequeñas y manejables.

A continuación se presentan un conjunto de clases y/o métodos que pueden resultar útiles en el proceso. Se recomienda consultar la documentación de las mismas para determinar qué papel pueden jugar en la arquitectura que usted propone.

```
DatagramPacket

AudioSystem.getAudioInputStream()

AudioSystem.getSourceDataLine()

ByteBuffer  .get()  y .put()

AudioInputStream   .getAudioInputStream()   y .read()

AudioFormat

SourceDataLine

BlockingQueue
```

Interpretar las siguientes porciones de código

```
// metodo para enviar un bloque de datos del lado del servidor
public static void sendAudio(String ip, int port, byte[] audioData, DatagramSocket socket)
throws Exception {
        InetAddress address = InetAddress.getByName(ip);
        DatagramPacket packet = new DatagramPacket(audioData, audioData.length, address,
port);
        socket.send(packet);
    }


// recepcion de un bloque de datos, del lado del cliente

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

socket.receive(packet);

buffer = packet.getData();

ByteBuffer byteBuffer = ByteBuffer.wrap(buffer);

int packetCount = byteBuffer.getInt();
```

Finalmente, se adjunta una implementación de una clase que permite recibir y reproducir bloques de audio mediante un hilo**: PlayerThread.java**

Deben analizar el código y evaluar cómo pueden usarlo en su solución.