Ejercicio: Comparación de Tiempos de Ejecución de Multiplicación de Matrices

Parte1:

El objetivo de este ejercicio es comparar los tiempos de ejecución del proceso de multiplicación de dos matrices utilizando un algoritmo secuencial y otro algoritmo utilizando hilos con ThreadPool. La multiplicación de matrices es una operación intensiva en términos computacionales y puede beneficiarse de la paralelización.

1. Implementación del Algoritmo Secuencial:

- Crea una clase **SequentialMatrixMultiplication** que contenga un método para multiplicar matrices de manera secuencial. Secuencial se refiere a la forma como usted lo haría a mano, siguiendo el paso a paso para obtener el resultado.

```
public class SequentialMatrixMultiplication {
     public static void multiply(int[][] matrixA, int[][] matrixB) {
         // Implementación de la multiplicación secuencial
     }
 }
```

2. Implementación del Algoritmo con ThreadPool:

- Crea una clase **ThreadPoolMatrixMultiplication** que utilice un ThreadPool para dividir el trabajo de multiplicar matrices entre múltiples hilos. Utiliza la clase **ExecutorService** para manejar el ThreadPool.

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ThreadPoolMatrixMultiplication {
    public static void multiply(int[][] matrixA, int[][] matrixB, int numThreads)
{
           ExecutorService executor = Executors.newFixedThreadPool(numThreads);
           // Implementación de la multiplicación con ThreadPool
           executor.shutdown();
           try {
               executor.awaitTermination(Long.MAX_VALUE, TimeUnit.NANOSECONDS);
           } catch (InterruptedException e) {
               e.printStackTrace();
           }
       }
   }
```

3. Medición de Tiempos de Ejecución:

- Implementa un método en cada clase para medir el tiempo de ejecución de la multiplicación de matrices.

```
public static long measureExecutionTime(Runnable task) {
       long startTime = System.nanoTime();
```

```
        task.run();
        long endTime = System.nanoTime();
        return endTime - startTime;
    }
```

4. Generación de Matrices Aleatorias:

- Crea un método para generar matrices aleatorias de tamaño adecuado.

```
import java.util.Random;
public class MatrixGenerator {
 public static int[][] generateRandomMatrix(int rows, int columns)
{
            int[][] matrix = new int[rows][columns];
            Random random = new Random();
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < columns; j++) {
                    // Números aleatorios entre 0 y 99
                    matrix[i][j] = random.nextInt(100);
                }
            }
            return matrix;
        }
    }
```

- Utiliza matrices de tamaño moderado para la comparación de tiempos, por ejemplo, matrices de 100x100 o 1000x1000.
- Se recomienda el uso de la interfaz **Runnable** para representar el trabajo que realizarán los hilos en el ThreadPool.
- Recuerda que la correcta gestión de recursos, como cerrar el ThreadPool después de su uso, es importante para evitar pérdidas de memoria.

Puntos a tener en cuenta:

División de la Tarea:

- Para multiplicar dos matrices, la tarea se puede dividir en pasos más pequeños. Por ejemplo, consideremos dos matrices, A y B, donde A es de tamaño *m* x *n* y B es de tamaño *n* x *p*. La multiplicación de matrices implica calcular el producto de cada fila de A con cada columna de B y sumar los resultados para obtener los elementos de la matriz resultante.

Para lograr esto, podemos dividir el trabajo en unidades más pequeñas de la siguiente manera:

Para cada fila i de la matriz A (donde i va desde 0 hasta m-1):

- Multiplicar la fila i de A por cada columna j de la matriz B (donde j va desde 0 hasta p-1).

Esta división de tareas garantiza que cada hilo se encargue de calcular una parte específica de la matriz resultante. Cada hilo sería responsable de calcular los elementos de una fila particular de la matriz resultante.

Por ejemplo, si tenemos una matriz A de 3x2 y una matriz B de 2x4, podríamos dividir el trabajo de la siguiente manera:

- Hilo 1: Calcula los elementos de la fila 0 de la matriz resultante.
- Hilo 2: Calcula los elementos de la fila 1 de la matriz resultante.
- Hilo 3: Calcula los elementos de la fila 2 de la matriz resultante (si la matriz resultante es de tamaño 3x4).

Cada hilo está encargado de multiplicar una fila específica de la matriz A por todas las columnas de la matriz B y sumar los resultados para obtener los elementos de su fila correspondiente en la matriz resultante.

Esta división de tareas asegura que cada hilo tenga una tarea claramente definida y que el trabajo total se distribuya eficientemente entre los hilos para aprovechar la paralelización y mejorar el rendimiento del cálculo.

Asignación de Tareas a Hilos:
Una vez que hemos identificado las tareas individuales, creamos hilos para ejecutar estas tareas. Por ejemplo, podríamos crear un hilo para cada fila de la matriz resultante, donde cada hilo sería responsable de calcular los elementos de esa fila.

Sincronización de Resultados:
Después de que cada hilo complete su tarea asignada, los resultados parciales deben ser combinados para obtener la matriz resultante completa. Dependiendo de cómo se dividió el trabajo, esto puede implicar la agregación de resultados parciales de cada hilo o la combinación de matrices parciales.

Manejo de Concurrencia:
Es importante tener en cuenta la sincronización y la concurrencia al dividir el trabajo entre los hilos. Los hilos deben acceder y modificar los datos compartidos (como la matriz resultante) de manera segura para evitar condiciones de carrera y otros problemas de concurrencia.

Gestión de Hilos:
Además, es importante gestionar adecuadamente los hilos, incluida la creación, ejecución y finalización de los hilos.

Parte 2: Modificar el ejemplo proporcionado para grabar audio de tal forma que permita grabar audio hasta que se presione una tecla. Por ejemplo, inicia la grabación y el usuario la termina presionando la tecla espacio.