**PRACTICA 8 |ARREGLOS MULTIDIMENSIONALES**

# Introducción

Estamos continuando con la tendencia del manejo de arreglos multidimensionales. Tanto la práctica pasada como la actual en sus instrucciones, indican que se está pretendiendo hacer lo que se hizo con las clases **Arreglo** y **VectorNum**, pero en su versión de dos dimensiones.

# Como lo comenté en la práctica pasada, estoy comenzando a sentir que mis habilidades en cuanto al manejo de estas estructuras de datos están mejorando. Espero poder realizar de una manera óptima y eficiente este ejercicio, y en su caso ir mejorando aún más, en su mayor parte a base de los errores que pudiese llegar a cometer.

# Desarrollo

La codificación de la práctica se tornó ágil en la mayoría de los métodos solicitados. Más de la mitad de las funciones fueron procedimientos relativamente fáciles de hacer que fueron implementados rápidamente. El único método que realmente me detuvo más tiempo y me hizo batallar, fue el que pedía realizar una multiplicación de matrices, el cual, su solución fue facilitada gracias a un análisis gráfico que tuve que hacer al ver que no funcionaba correctamente.

Por otra parte, en los últimos dos métodos esperaba tener más complicaciones, a la hora de formular una condición que me permitiera detectar la matriz triangular inferior o superior en cada caso, aquí, a diferencia de la multiplicación de matrices, sí realicé realmente el análisis previo correspondiente y eso se reflejó en un tiempo de solución muy favorable.

Finalmente, en esta práctica fue creada una nueva clase de nombre **MatrizNum**,que hereda de **Matriz**,y otra de tipo enum llamada **TipoLogaritmo**, cada una con sus métodos correspondientes.

A continuación, se indican los archivos creados o modificados y sus métodos.

## MatrizNum.java (paquete edlineal).

* public boolean setElemento(int reng, int col, Object valor).
* public inicializar(Object valor).
* private boolean validarElementos(Matriz matriz).
* private boolean validarElementos(Arreglo arreglo).
* public boolean esIgual(Matriz matriz2).
* public boolean vectorCol(int numRenglones, Object valor).
* public boolean vectorReng(int numColumnas, Object valor).
* public boolean definirMatriz(Matriz matriz2).
* public boolean agregarRenglon(Arreglo arreglo).
* public boolean agregarColumna(Arreglo arreglo).
* public boolean agregarMatrizColumna(Matriz matriz2).
* public boolean agregarMatrizRenglon(Matriz matriz2).
* public Matriz3D aMatriz3D(Arreglo matrices).
* public void multiplicarEscalar(double escalar).
* public boolean multiplicarMatriz(Matriz matriz2).
* public boolean sumarMatriz(Matriz matriz2).
* public void aplicarPotencia(double escalar).
* public boolean aplicarLog(TipoLogaritmo tipoLog).
* public void matrizDiagonal(Object valor).
* public boolean esDiagonalSup().
* public boolean esDiagonalInf().

**Los métodos subrayados han sido sobrescritos de la clase Matriz, para validar que los elementos que se agregan o se asignan sean instancias de Number.**

## TipoLogaritmo.java (paquete matematicas).

* public String getNombre().
* public int getId().

**Se incluyeron dos atributos para este enum: “BASE10” y “NATURAL”.**

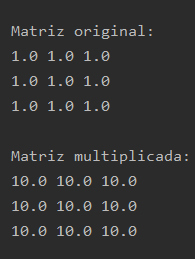
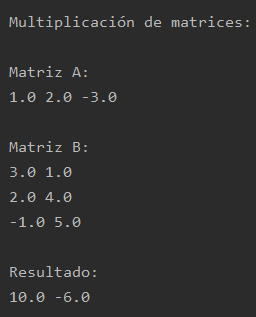
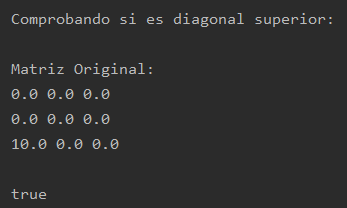
## PruebaMatrizNum.java (paquete interfaces).

La clase anterior es donde se encuentran las pruebas de esta práctica.

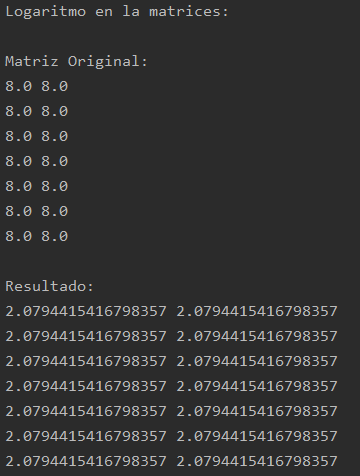
**Para mayor información sobre los métodos y la clase creada vaya a la ruta: /docs/index.html dentro del proyecto.**

# Capturas de pantalla del programa funcionando

**Multiplicación por un escalar. Multiplicación de matrices.** **Matriz triangular superior.**

 ****

**Logaritmo de matriz.**

****

# Conclusiones

Nuevamente tuve que acatar la lección de no haber hecho un análisis previo, sobre el papel para el caso de la multiplicación de matrices, cosa que me detuvo más de lo que esperaba por estar pensando cómo solucionarlo directamente desde el código. Efectivamente hubo un poco más de mejoría en el manejo de matrices en esta práctica, y espero para las demás prácticas ya no volver a minimizar un problema (a menos que de verdad lo sea) y realizar siempre lo que sea necesario para encontrar una solución antes de codificar.