



UNIVERSIDAD AUTÓNOMA DE ZACATECAS

Practica

Estudiante:

José Francisco Hurtado Muro

Profesor:

Dr. Aldonso Becerra Sánchez

Tabla de Contenidos

1	Introducción	2
2	Actividades	3
2.1	Actividad inicial:	3
2.2	Actividad 1:	3
2.3	Actividad 2:	3
2.3.1	Constructor: Arreglo2D (int renglones, int columnas)	3
2.3.2	Constructor: Arreglo2D (int renglones, int columnas, Object valor)	3
2.3.3	void llenar(Object valor)	3
2.3.4	Object obtener(int fila, int col)	4
2.3.5	boolean cambiar(int fila, int col, Object valor)	5
2.3.6	void imprimirXColumnas()	5
2.3.7	void imprimirXRenglones()	6
2.3.8	Arreglo2D clonar()	6
2.3.9	boolean esIgual(Arreglo2D matriz2)	7
2.3.10	boolean vectorRenglon(int columnas, Object info)	8
2.3.11	boolean redefinir(Arreglo2D matriz2)	9
2.3.12	boolean agregarRenglon(Arreglo arreglo)	9
2.3.13	boolean agregarColumna(Arreglo arreglo)	10
2.3.14	boolean vectorColumna(int filas, Object info)	11
3	Código Agregado - UML	12
4	Pre-evaluación del Alumno	13
5	Conclusión	13
6	Referencias:	13

1 Introducción

"La facilidad que los arreglos multidimensionales tienen para permitir guardar datos en forma de columnas/renglones, los hace pertinentes para la resolución de muchos problemas donde se requiere esta situación. El único detalle con esta cuestión es que es poco flexible el número de elementos que podemos manipular, ya que se requiere conocer a priori la cantidad de elementos a guardar."

2 Actividades

2.1 Actividad inicial:

Lea primero toda la práctica. No inicie a programar sin leer todo cuidadosamente primero. Recuerde que debe generar el reporte en formato IDC.

2.2 Actividad 1:

Primero genere la **Introducción**.

2.3 Actividad 2:

Defina un TDA llamado “Arreglo2D”, que permita realizar las siguientes operaciones:

2.3.1 Constructor: Arreglo2D (int renglones, int columnas)

Explicación:

Para este caso del constructor, lo que hace es iniciar el arreglo en dos dimensiones con un tamaño fijo, que recibe cuantos renglones y columnas se quieren, teniendo en cuenta de que el arreglo2D inicia en cero.

2.3.2 Constructor: Arreglo2D (int renglones, int columnas, Object valor)

Explicación:

En este caso es muy parecido en el anterior **Arreglo2D (int renglones, int columnas)**, pero a diferencia del anterior, este lo que hace es que inicia todas las posiciones del arreglo2D con un valor datos. en la figura 1.

Para este metodo se ocupo de ayuda de **void llenar(Object valor)** ya qyu esta metodo ya realiza la tarea de llenar todo el arreglo2D.

2.3.3 void llenar(Object valor)

Explicación:

 Inicializar todos los elementos de la matriz

Este caso lo que hace es que recibe un valor valor el cual sera puesto en todos en todas las posiciones del arreglo2D, esto lo hace por medio de ciclos, un primero para recorrer las columnas, y un segundo recorriendo lo que son las filas, y es aquí que va poniendo el elemento dado en todos las posiciones

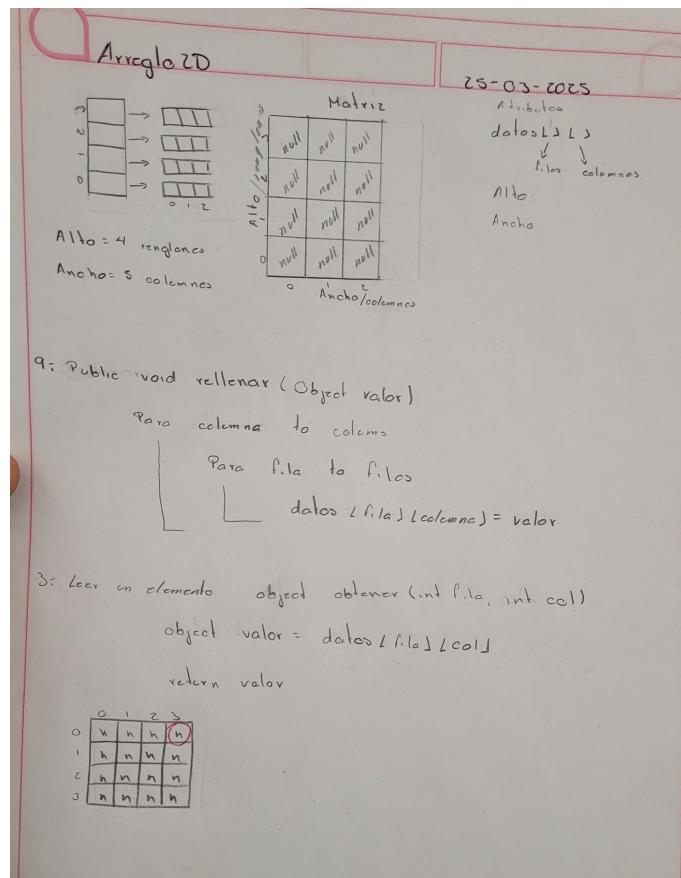
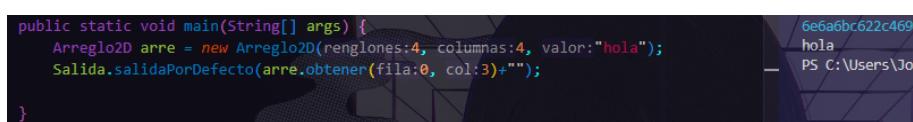


Figure 1: Análisis: Constructores, obtener y llenar

2.3.4 Object obtener(int fila, int col)

Explicación: Leer un elemento de la matriz

Aquí lo que se realizo fue de los valores dados, fila y col, estos valores representan posiciones dentro del arreglo de donde se sacara el dato electo



```
public static void main(String[] args) {
    Arreglo2D arre = new Arreglo2D(renglones:4, columnas:4, valor:"hola");
    Salida.salidaPorDefecto(arre.obtener(fila:0, col:3),"");
}
```

6e6a6bc622c469
hola
PS C:\Users\Jo

Figure 2: leyenda de imagen-captura

2.3.5 boolean cambiar(int fila, int col, Object valor)

Cambiar un elemento.

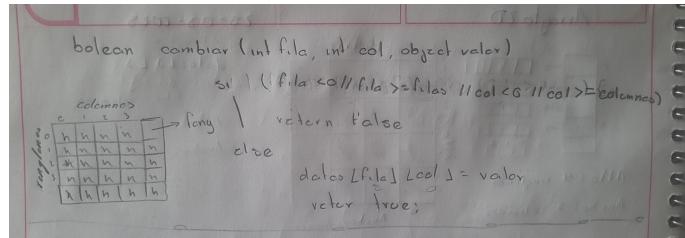


Figure 3: Análisis: cambiar elemento

Explicación:

En este caso lo que se realizó fue que a partir de dos valores, fila y col, podemos modificar un valor en una posición y remplazarlo por un valor que le digamos, para esto tuvimos que validar que los dos valores numéricos que representan las ordenadas, estén dentro del rango de los valores del arreglo2D, en la figura 4 podemos ver su funcionamiento.



```

Run | Debug
public static void main(String[] args) {
    Arreglo2D arre = new Arreglo2D(4, 4, "holas");
    arre.cambiar(fila:0, col:3, valor:"Estefania");
    Salida.salidaPorDefecto(arre.obtener(fila:0, col:3)
) + "");
    
```

ae96e6a6bc622c46:
Estefania
PS C:\Users\Jose

Figure 4: Funcionamiento: cambiar elemento

2.3.6 void imprimirXColumnas()

Imprimir los datos de la matriz: void imprimirXColumnas(). Imprime la matriz en formato columna por columna.

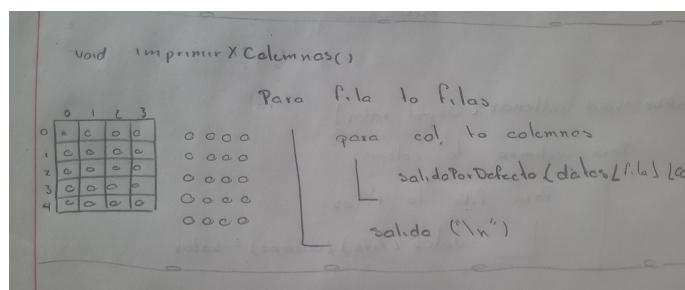


Figure 5: Análisis: imprimirXColumnas

Explicación:

Para este caso en particular se hizo un recorrido sobre cada fila, pero antes de recorrer la siguiente fila, se recorre sobre todos los valores de las columnas, y con esto se va imprimiendo todos los valores, y al terminar el recorrido de la columna se hace un salto de linea, en la figura 6 podemos ver el funcionamiento.



```

Run | Debug
public static void main(String[] args) {
    Arreglo2D arre = new Arreglo2D(renglones:4, columnas:3, valor:"0");
    arre.cambiar(fila:0, col:2, valor:"f");
    //Salida.salidaPorDefecto(arre.obtener(0, 2)+"");
    arre.imprimirXColumnas();
}
    
```

aceStorage
tica6Arr
00f
000
000
000
PS C:\User

Figure 6: Funcionamiento: imprimirXColumnas

2.3.7 void imprimirXRenglones()

Imprimir los datos de la matriz: void imprimirXRenglones(). Imprime la matriz en formato renglón por renglón.

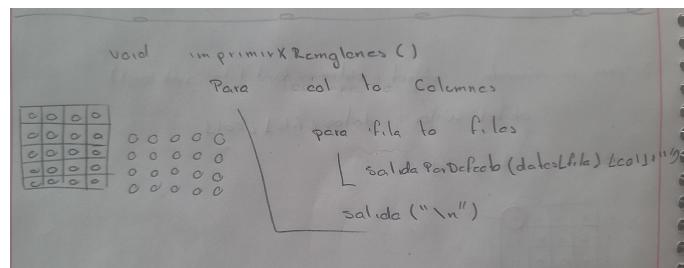


Figure 7: Análisis: imprimirXRenglones

Explicación:

Este caso es muy similar al anterior, lo que se hizo de diferente fu invertir el ordrn de los ciclos, primero es es que recorre el arreglo de columnas y luego el arreglo de filas, y este va imprimiendo los valores de las filas como podemos ver en la figura 8.



```

Run | Debug
public static void main(String[] args) {
    Arreglo2D arre = new Arreglo2D(renglones:4, columnas:3, valor:"0");
    arre.cambiar(fila:0, col:2, valor:"f");
    //Salida.salidaPorDefecto(arre.obtener(0, 2)+"");
    arre.imprimirXRenglones();
}
    
```

aceStorage
tica6Arr
000
000
f000
PS C:\User

Figure 8: Funcionamiento: imprimirXRenglones

2.3.8 Arreglo2D clonar()

Generar y regresar una “copia” de la matriz: Arreglo2D clonar().

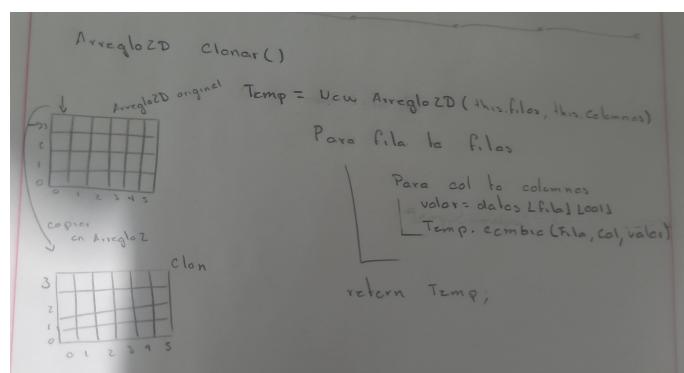


Figure 9: Análisis: clonar

Explicación:

Aquí lo que se hizo fue crear un arreglo2D temporal, ya que este con un funcionamiento muy parecido a **imprimirXColumnas**, con la diferencia de que en vez de imprimir lo que hace es guardar los elementos del arreglo original en el temporal, una vez con todos los elementos ya en el arreglo2d temporal, lo retornamos y debería de verse como en la figura 10.



```

Run | Debug
public static void main(String[] args) {
    Arreglo2D arre = new Arreglo2D(renglones:4, columnas:3)
    arre.cambiar(fila:0, col:2, valor:"f");
    //Salida.salidaPorDefecto(arre.obtener(0, 2)+"\n");
    Salida.salidaPorDefecto(cadena:"original\n");
    Arreglo2D clone = arre.clonar();
    arre.imprimirXColumnas();
    Salida.salidaPorDefecto(cadena:"clon\n");
    clone.imprimirXColumnas();
}

```

Figure 10: Funcionamiento: clonar

2.3.9 boolean esIgual(Arreglo2D matriz2)

Generar un método que diga si el contenido de dos matrices es igual: boolean esIgual(Arreglo2D matriz2). Posición por posición como contenido.

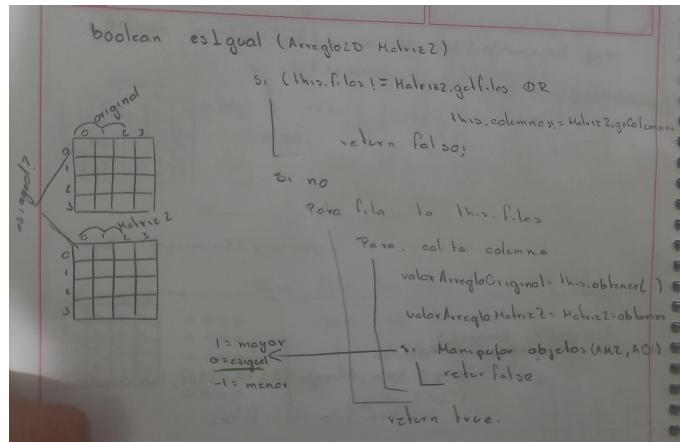
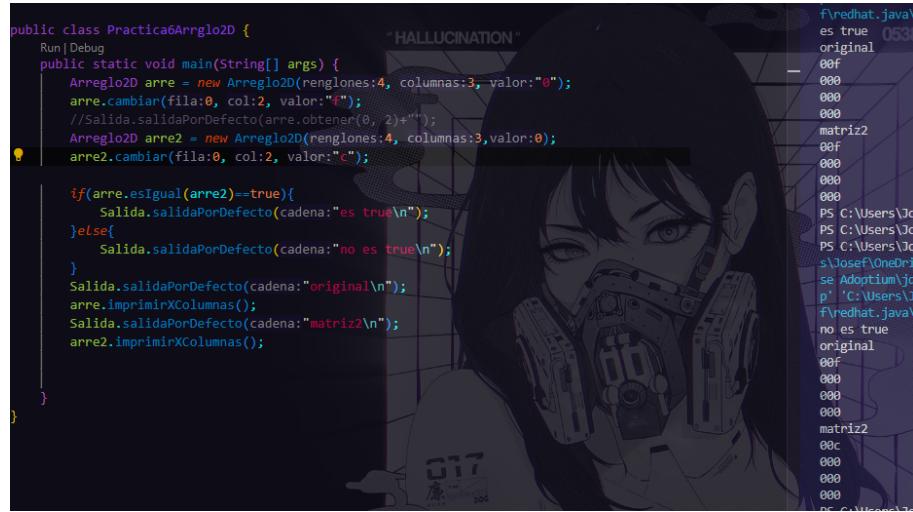


Figure 11: Análisis: esIgual

Explicación:

Aquí lo que se hizo fui validar el tamaño de las filas y columnas de ambas matrices, si una no coincidía con la otra, automáticamente se regresa un false, pero si pasa esta validación, lo siguiente seria validar los valores de las posiciones, se uso un método ya fabricado en otra practica llamado **ManipularObjetos.compararObjetos**, que retorna 3 posibles valores, 1 si el valor es mayor, -1 si el valor es menor y 0 si el valor es igual, lo cual nos enfocamos en este ultimo donde mientras no de un valor diferente a 0 significa que los elementos de la matriz son iguales a los de la matriz2, si llegase a ser diferente de 0 regresa false, una vez pasado todos lo elementos por esta validacion y si todos fueron 0, podemos regresar un true de que eran iguales las dos matrices, como podemos ver en la figura 12.



```

public class Practica6Arreglo2D {
    public static void main(String[] args) {
        Arreglo2D arre = new Arreglo2D(renglones:4, columnas:3, valor:"0");
        arre.cambiar(fila:0, col:2, valor:"f");
        //Salida.salidaPorDefecto(arre.obtener(0, 2)+"");
        Arreglo2D arre2 = new Arreglo2D(renglones:4, columnas:3, valor:0);
        arre2.cambiar(fila:0, col:2, valor:"c");

        if(arre.esIgual(arre2)==true){
            Salida.salidaPorDefecto(cadena:"es true\n");
        }else{
            Salida.salidaPorDefecto(cadena:"no es true\n");
        }
        Salida.salidaPorDefecto(cadena:"original\n");
        arre.imprimirXColumnas();
        Salida.salidaPorDefecto(cadena:"matriz2\n");
        arre2.imprimirXColumnas();

    }
}
    
```

Figure 12: Funcionamiento: esIgual

2.3.10 boolean vectorRenglon(int columnas, Object info)

Defina un método que genere un vector renglón con los datos especificados por el usuario, a partir de la matriz vacía: boolean vectorRenglon(int columnas, Object info).

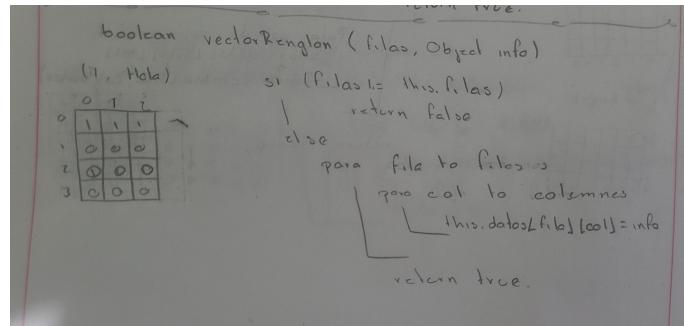


Figure 13: Análisis: vectorRenglon

Explicación:

Lo que se hizo en este metodo fue iterar sobre la cantidad de filas dado el valor filas y en cada columna ir agregando el valor info que tambien se proporciono, y se debería de ver como en la figura 14.



```

if(arre.vectorRenglon(filas:0, info:1)){ "HALLUCINATION"
    Salida.salidaPorDefecto(cadena:"es true\n");
}else{
    Salida.salidaPorDefecto(cadena:"no es true\n");
}

arre.imprimirXColumnas();
    
```

Figure 14: Funcionamiento: vectorRenglon

2.3.11 boolean redefinir(Arreglo2D matriz2)

Defina un método que permita crear/redimensionar/substituir la tabla actual por una pasada como argumento. boolean redefinir(Arreglo2D matriz2).

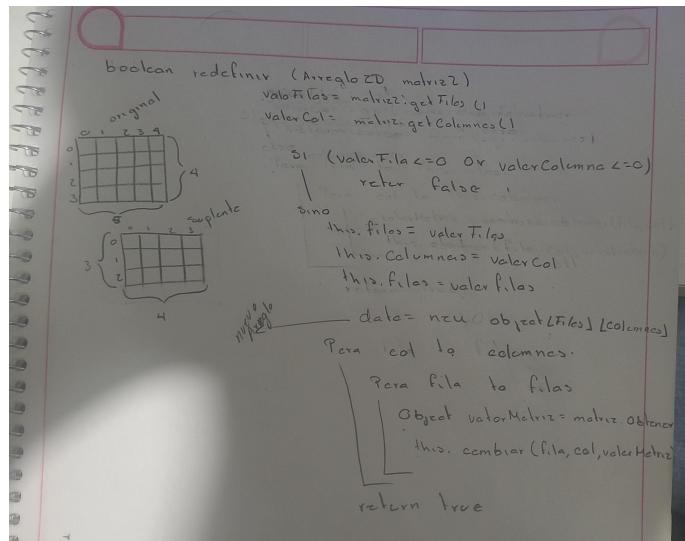


Figure 15: Análisis: redefinir

Explicación:

Para este caso lo que se hizo fu recibir un arreglo2D al cual le sacamos sus dimensiones y validamos que no sean cero, si pasa esta validación lo que pasaría es que se cambiaran los valores de filas y columnas por los obtenidos, ademas de que al arreglo datos se le instancia un nuevo arreglo, para esto solo queda poner los datos de la matriz dada en la original pero ya con los datos de la segunda matriz dada, ademas de eso que y se debería de ver como en la figura 16.

```

Arreglo2D arre2 = new Arreglo2D(renglones:3, columnas:10, valc
Salida.salidaPorDefecto(cadena:"arreglo original\n");
arre.imprimirXColumnas();

if(arre.redefinir(arre2)){
    Salida.salidaPorDefecto(cadena:"es true\n");
} else{
    Salida.salidaPorDefecto(cadena:"no es true\n");
}

Salida.salidaPorDefecto(cadena:"arreglo final\n");
arre2.imprimirXColumnas();
    
```

Figure 16: Funcionamiento: redefinir

2.3.12 boolean agregarRenglón(Arreglo arreglo)

Defina un método que permita agregar una Arreglo pasado como argumento como renglón de la matriz existente. Para esto deberá validar los tamaños del arreglo que cumplan con el cometido de la matriz. boolean agregarRenglón(Arreglo arreglo).

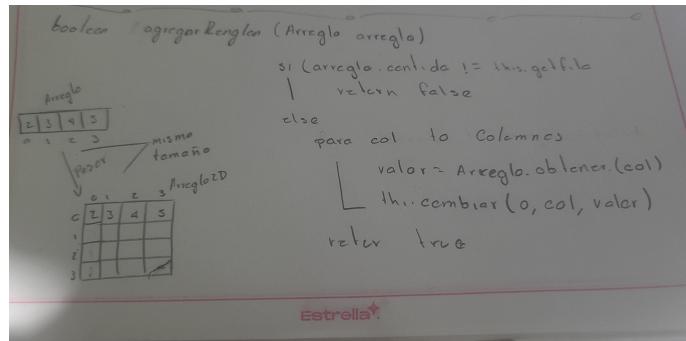
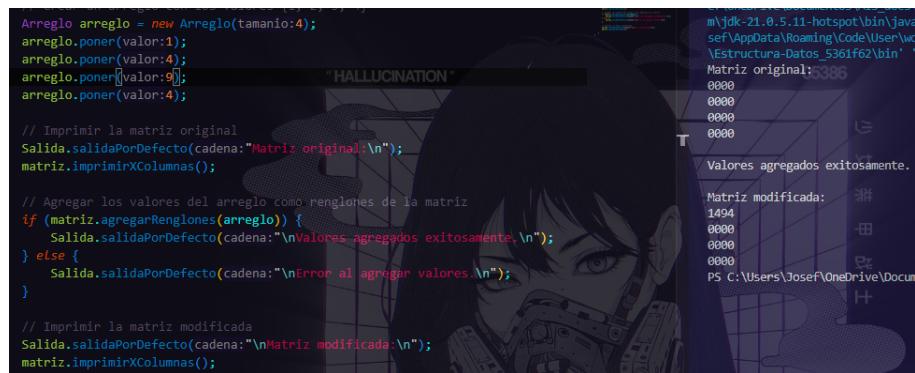


Figure 17: Análisis: agregarRenglon

Explicación:

En este caso lo que se hizo fue recibir un Arreglo, donde le extraeremos cada elemento y lo iremos agregando a la matriz, pero solo en la primera posición, como podemos apreciar en la figura 18.



```

Arreglo arreglo = new Arreglo(tamano:4);
arreglo.poner(valor:1);
arreglo.poner(valor:4);
arreglo.poner(valor:9);
arreglo.poner(valor:4);

// Imprimir la matriz original
Salida.salidaPorDefecto(cadena:"Matriz original:\n");
matriz.imprimirXColumnas();

// Agregar los valores del arreglo como renglones de la matriz
if (matriz.agregarRenglones(arreglo)) {
    Salida.salidaPorDefecto(cadena:"\nValores agregados exitosamente.\n");
} else {
    Salida.salidaPorDefecto(cadena:"\nError al agregar valores.\n");
}

// Imprimir la matriz modificada
Salida.salidaPorDefecto(cadena:"\nMatriz modificada:\n");
matriz.imprimirXColumnas();
    
```

Figure 18: Funcionamiento: agregarRenglon

2.3.13 boolean agregarColumna(Arreglo arreglo)

Defina un método que permita agregar una Arreglo pasada como argumento como columna de la matriz existente. Para esto deberá validar los tamaños del arreglo que cumplan con el cometido de la matriz. boolean agregarColumna(Arreglo arreglo).

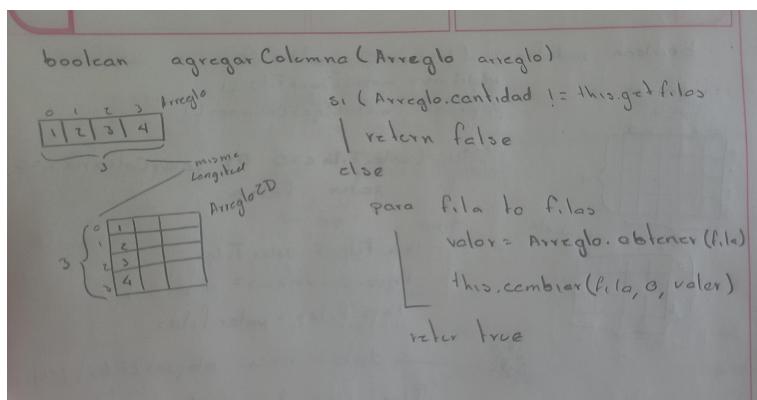


Figure 19: Análisis: agregarColumna

Explicación:

Este caso es muy similar al anteriores, con la diferencia que se pongan en columna, por lo cual en ves de iterar sobre las columnas iteran sobre las filas y va agregando los datos en los espacio necesarios, en la figura 20 podemos ver su funcionamiento.



```

arreglo.poner(valor:4);
arreglo.poner(valor:9);
arreglo.poner(valor:4);

// Imprimir la matriz original
Salida.salidaPorDefecto(cadena:"Matriz original:\n");
matriz.imprimirXColumnas();

// Agregar los valores del arreglo como renglones de la matriz
if (matriz.agregarColumna(arreglo)) {
    Salida.salidaPorDefecto(cadena:"\nValores agregados exitosamente.\n");
} else {
    Salida.salidaPorDefecto(cadena:"\nError al agregar valores.\n");
}

// Imprimir la matriz modificada
Salida.salidaPorDefecto(cadena:"\nMatriz modificada:\n");
matriz.imprimirXColumnas();

```

The terminal output shows the original matrix (4x4 with all zeros), the addition of the values [4, 9, 4] as new rows, and the resulting modified matrix (4x5).

Figure 20: Funcionamiento: agregarColumna

2.3.14 boolean vectorColumna(int filas, Object info)

DDefina un método que genere un vector columna con los datos especificados por el usuario, a partir de la matriz vacía: boolean vectorColumna(int filas, Object info).

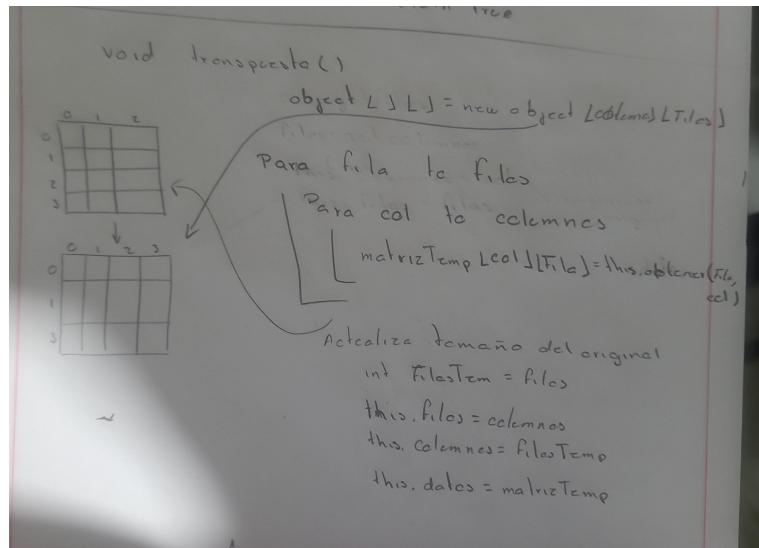


Figure 21: Análisis: vectorColumna

Explicación:

Este caso es muy parecido al caso **vectorRenglon**, con la diferencia de que los datos deben ser insertados de manera vertical, donde lo que se hizo evaluar el rango de filas que no se saliera del rango permitido, al pasar esa evaluación se recorre entre filas para después de recorrer el numero de columnas deseadas por el usuario y ir añadiendo el valor info, tal y como podemos ver en la figura 22.



```

Arreglo2D matriz = new Arreglo2D(renglones:3, columnas:2);
matriz.cambiar(fila:0, col:0, valor:1);
matriz.cambiar(fila:0, col:1, valor:2);
matriz.cambiar(fila:1, col:0, valor:3);
matriz.cambiar(fila:1, col:1, valor:4);
matriz.cambiar(fila:2, col:0, valor:5);
matriz.cambiar(fila:2, col:1, valor:6);

// Imprimir la matriz original
Salida.salidaPorDefecto(cadena:"Matriz original:\n");
matriz.imprimirXColumnas();

// Realizar la transposición
matriz.transpuesta();

// Imprimir la matriz transpuesta
Salida.salidaPorDefecto(cadena:"\nMatriz transpuesta:\n");
matriz.imprimirXColumnas();

```

Figure 22: Funcionamiento: vectorColumna

3 Código Agregado - UML

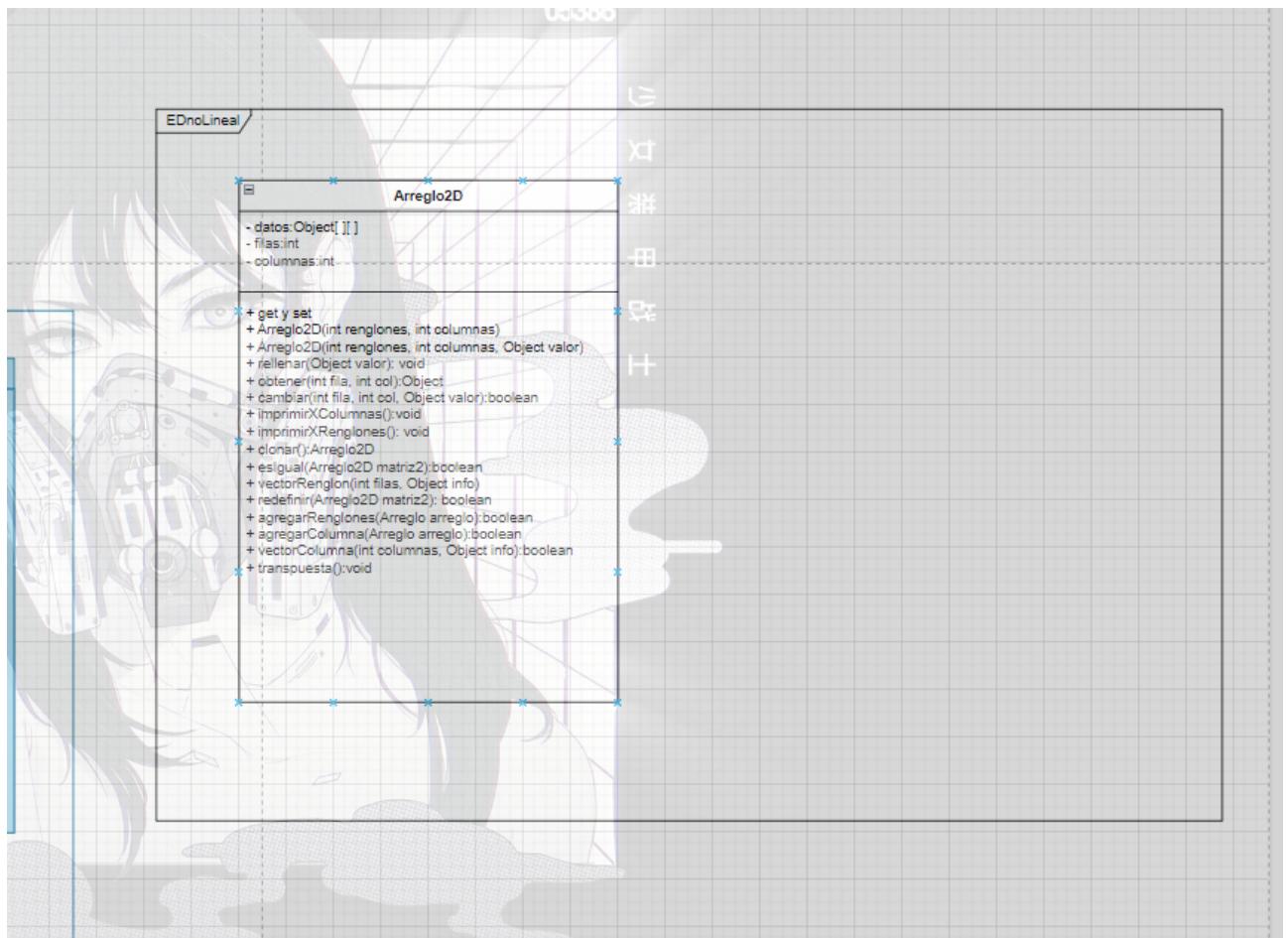


Figure 23: UML

4 Pre-evaluación del Alumno

Criterio	Evaluación
Cumple con la funcionalidad solicitada	Sí
Dispone de código auto-documentado	Sí
Dispone de código documentado a nivel de clase y método	Sí
Dispone de indentación correcta	Sí
Cumple la POO	Sí
Dispone de una forma fácil de utilizar el programa para el usuario	Sí
Dispone de un reporte con formato IDC	Sí
La información del reporte está libre de errores de ortografía	Sí
Se entregó en tiempo y forma la práctica	No
Incluye el código agregado en formato UML	Sí
Incluye las capturas de pantalla del programa funcionando	Sí
La práctica está totalmente realizada (especifique el porcentaje completado)	78%

Table 1: Evaluación de la práctica

5 Conclusión

Se trabajo de manera correcta con arreglos 2D los cuales me parecieron curiosos pero útiles.

6 Referencias:

- Cairo, Osvaldo; Guardati, Silvia. *Estructura de Datos, Tercera Edición*. McGraw-Hill, México, Tercera Edición, 2006.
- Mark Allen Weiss. *Estructura de datos en Java*. Ed. Addison Wesley.
- Joyanes Aguilar, Luis. *Fundamentos de Programación. Algoritmos y Estructuras de Datos*. Tercera Edición, 2003. McGraw-Hill.