Universidad Autónoma de Zacatecas

Practica 7: Arreglos multidimensionales

Estudiante:
José Francisco Hurtado Muro

Profesor:
Dr. Aldonso Becerra Sánchez

March 30, 2025

Tabla de Contenidos

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 dónde 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 "Arreglo2DNumerico" que herede de "Arreglo2D", de modo que este tipo de matriz pueda realizar lo siguiente:

2.3.1 Validación de datos tipo Number

Valide que todos los valores pasados a cada método de esta clase sean estrictamente heredados de "Number", tal como se hizo en ArregloNumerico. Para esto deberá sobre-escribir e invocar los métodos de la super clase de tal manera que se cumpla este cometido también en esos métodos. Guarde los valores como double de manera predeterminada

```
@Override //rellenar la matriz con algun valor numerico
public void rellenar(Object valor){
    if(valor instanceof Number){ //validar que valor sea instancia de Number
```

Figure 1: Validación de instancia

Explicación
Para poder restringir el tipo de valores en los métodos existentes, se uso la función **instanceof** que lo que hace es revisar si es dato o objeto es instancia de una clase en especifico, o por medio de polimorfismo ver si es una clase hija de la clase a revisar, en este ejemplo podemos ver que podemos preguntar si un dato es de la clase Number, donde puede entrar todas aquellas clases hijas como: Double o Integer.
En la figura 1 podemos ver un ejemplo del uso de **instanceof**.

2.3.2 Constructor: Arreglo2DNumerico (int filas, int columnas)

Constructor: Arreglo2DNumerico (int filas, int columnas). Que llame a inicializar con valores en cero de manera predeterminada.

```
//inicia con un tamanio y valores en cero
public Arreglo2DNumerico(int renglones, int columnas){
    super(renglones, columnas); // le da un tamanio al arreglo2D
    this.rellenar(valor:0); // se rellena con cero
}
```

Figure 2: Arreglo2DNumerico (int filas, int columnas)

Explicación
Lo que se hizo en este fue llamar al constructor de la clase padre que solo recibe dos parámetros, filas y columnas, esto para que el arreglo solo se encargue de darle el tamaño necesario, una vez ya iniciado el arreglo2D, se le indica con la función **rellenar( )** que todos lo valores de este arreglo2D sean cero. En la figura 2 podemos ver esto de manera mas clara.

2.3.3 Constructor: Arreglo2DNumerico (int filas, int columnas, Object valor)

Constructor: Arreglo2DNumerico (int filas, int columnas, Object valor). Que llame a inicializar con valores indicados por contenido.

```
//inicia un arreglo 2D con un valor dado
public Arreglo2D(int renglones, int columnas, Object valor){
    this.filas = renglones;
    this.columnas = columnas;
    datos = new Object[filas][columnas];
    rellenar(valor);
}
```

Figure 3: Arreglo2DNumerico (int filas, int columnas, Object valor)

Explicación
Este caso es similar al anterior visto en la figura 2, con la diferencia, este lo que hace es que aparte de recibir el los valores del tamaño también recibe un valor de tipo Object, el cual sera nuestro valor con el que iniciaremos todo el arreglo2D como lo podemos ver en la figura 3

2.3.4 boolean porEscalar(Number escalar)

Multiplicar un escalar por la matriz: boolean porEscalar(Number escalar). Debe validar las dimensiones y el procedimiento de acuerdo a la teoría de matrices.

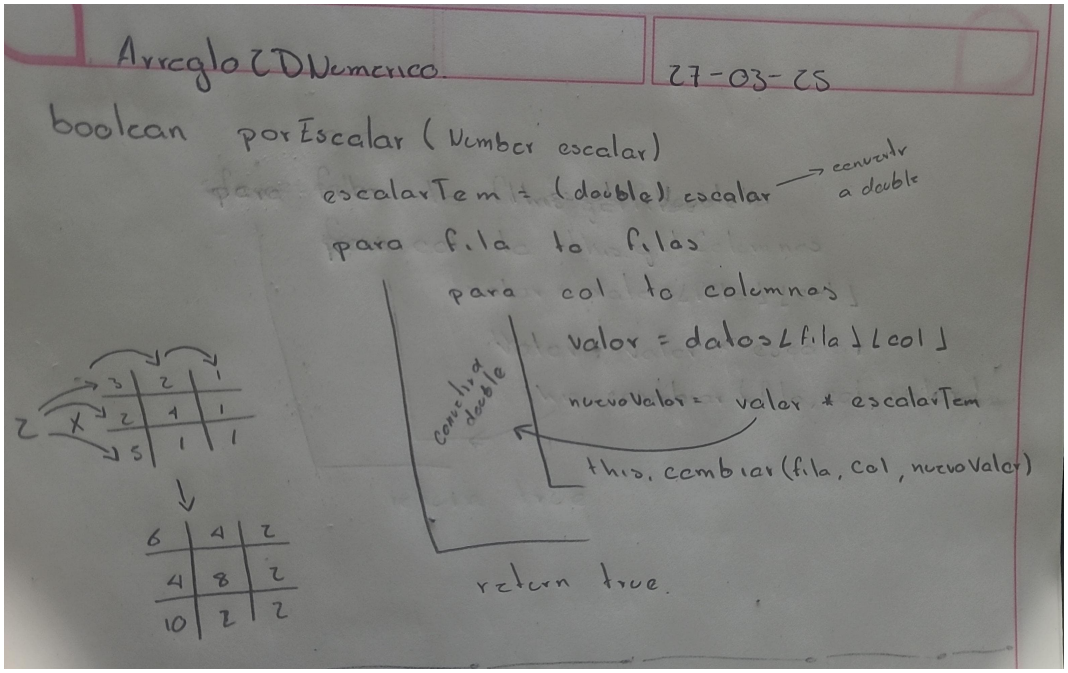


Figure 4: Análisis: porEscalar(Number escalar)

Explicación

Aquí lo que se hizo es que dado un escalar, este se le aplique a todos los elementos de una matriz, para esto al escalar paso de Number a ser tipo doouble, para poder usar operadores aritméticos, como siguiente se itero sobre cada posición en la matriz extrayendo el elemento y multiplicándolo por el escalar, una vez echo la multiplicación lo regresamos a la posición donde se extrajo el valor de la matriz como podemos ver en la figura 5.

Figure 5: Funcionamiento: porEscalar(Number escalar)

2.3.5 boolean porEscalares(ArregloNumerico escalares)

Multiplicar un escalar para cada elemento de la matriz: boolean porEscalares(ArregloNumerico escalares). Multiplica el elemento de una posición del arreglo de números (cada posición contiene un escalar) por un solo elemento colocado en la posición correspondiente en la matriz, recorriéndola en orden natural (renglones, columnas). Quiere decir que el escalar de la posición 0 del arreglo se multiplica por el elemento 0,0 de la matriz, el escalar de la posición 1 del arreglo se multiplica por el elemento 0,1 de la matriz, etc.

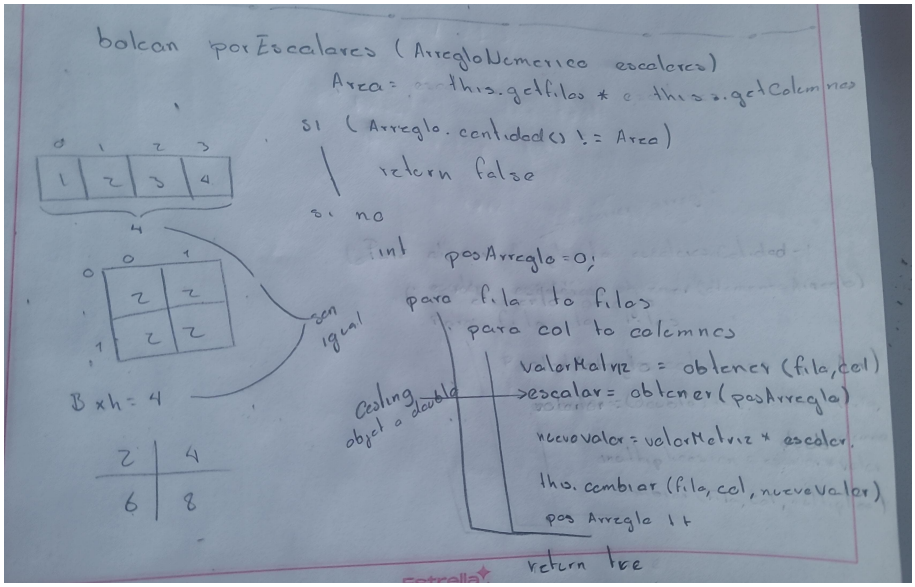


Figure 6: Análisis: porEscalares(ArregloNumerico escalares

Explicación

Aquí lo que se hizo fue recibir un arregloNumerico al cual compramos su longitud con el area de la matriz, esta estas no coinciden regresa un falso, pero si coinciden se inicia un acumulador de la posición actual en en arreglo, esto para llevar un contro de la variable que se va a sacar del arregloNumerico, se recorre el arreglo dos extrayendo elemnto por elemento, donde a cada uno de ellos le aplicamos la multiplicación por el escalar que coincida en la posición del arreglo2DNumerico como podemos ver en la figura 7.

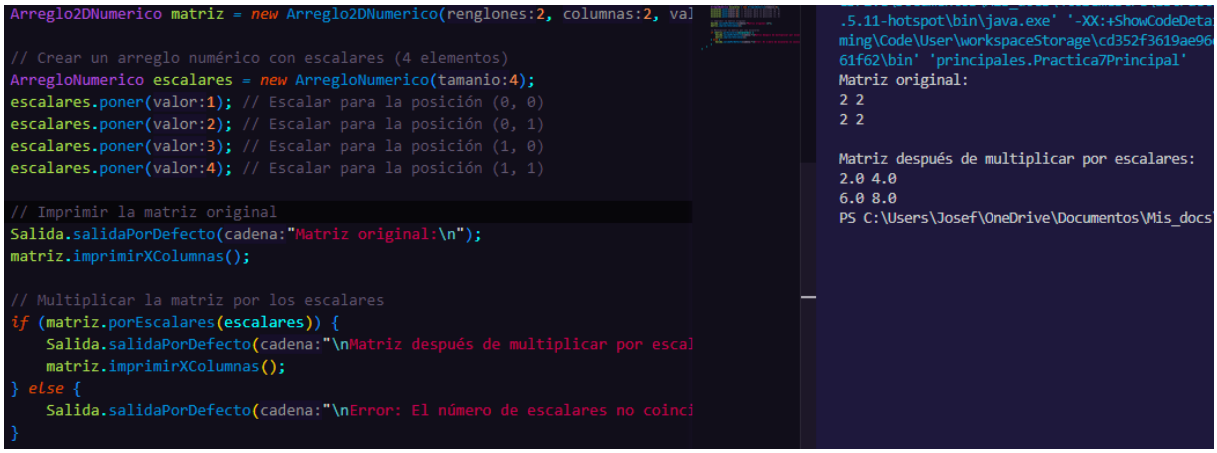


Figure 7: Funcionamiento: porEscalares(ArregloNumerico escalares)

2.3.6 boolean sumarEscalar(Number escalar)

Debe validar las dimensiones y el procedimiento de acuerdo a la teoría de matrices.

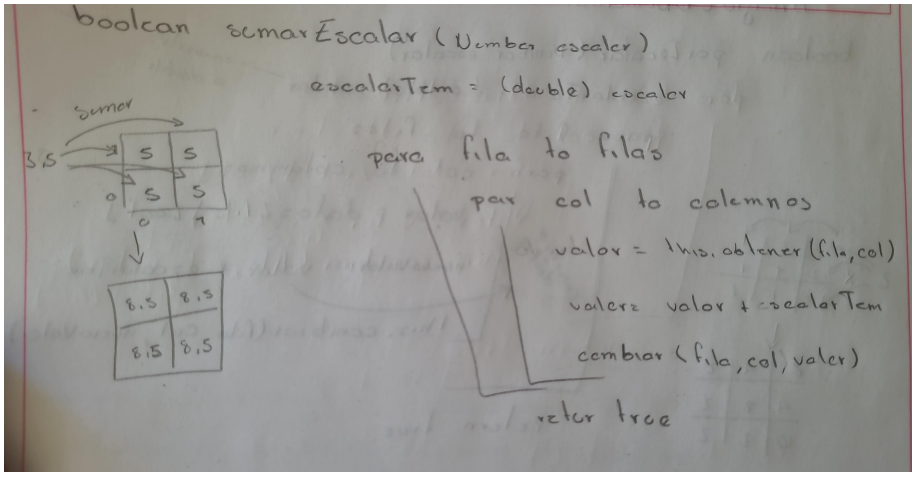


Figure 8: Análisis: sumarEscalar(Number escalar)

Explicación

En este caso en particular se ignoro la validascion de dimenciones, ya que una suma de escalar a una matriz no se puede, segun el algebra para hacer una suma lo que se debe hacer lo siguiente:

"Para sumar dos matrices, deben cumplir la regla principal:

- Ambas matrices deben tener las mismas dimensiones (es decir, el mismo número de filas y columnas).

Si tienes dos matrices A y B de tamaño $m \times n$, su suma C se define como:

$$C[i][j] = A[i][j] + B[i][j]$$

para cada posición (i, j) en la matriz."

pero en nuestro caso solo recibimos un valor único, no otra matriz, por lo cual tendríamos que crear una matriz con las mismas longitudes con un único valor, pero para esto podemos extraer cada valor de la matriz y solo sumarle el valor y ingresarlo a la misma posición ocomo podemos ver en la figura 8.

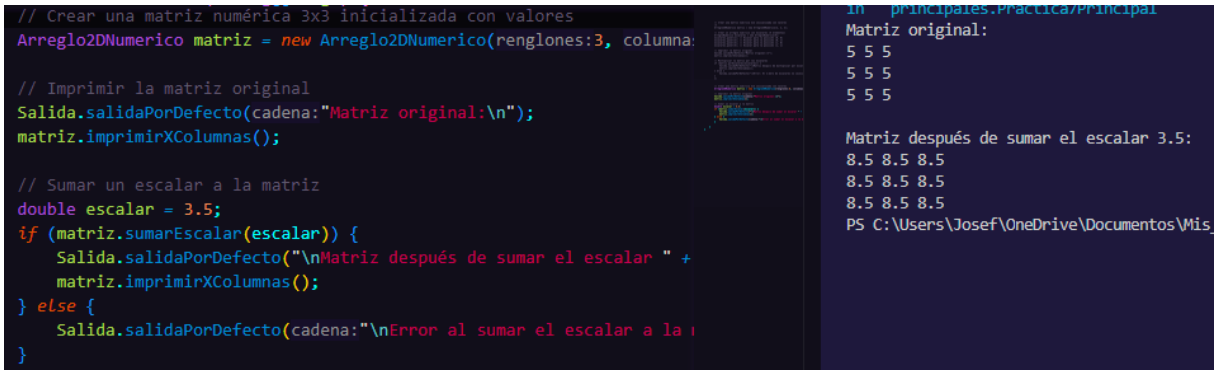


Figure 9: Funcionamiento: sumarEscalar(Number escalar)

2.3.7 boolen sumarEscalares(ArregloNumerico escalares)

Sumar un escalar para cada elemento de la matriz: boolean sumarEscalares(ArregloNumerico escalares). Suma un elemento contenido en una posición del arreglo de números (cada posición contiene un escalar) más un solo elemento colocado en la posición correspondiente en la matriz, re-corriéndola en orden natural (renglones, columnas). Quiere decir que el escalar de la posición 0 del arreglo se suma al elemento 0,0 de la matriz, el escalar de la posición 1 del arreglo se suma al elemento 0,1 de la matriz, etc

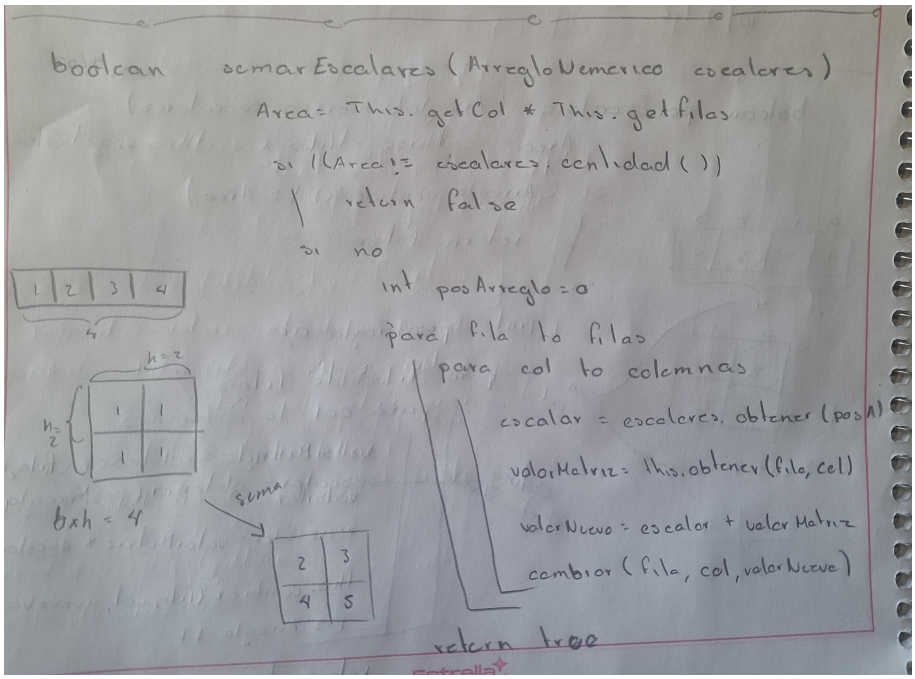


Figure 10: Análisis: sumarEscalares(ArregloNumerico escalares)

Explicación

Este caso es muy parecido al método **porEscalares**, con la diferencia, de que en ves de multiplicar las posicisiones, se suman, y se vuelve a insertar en la misma posición como podemos ver en la figura 11.

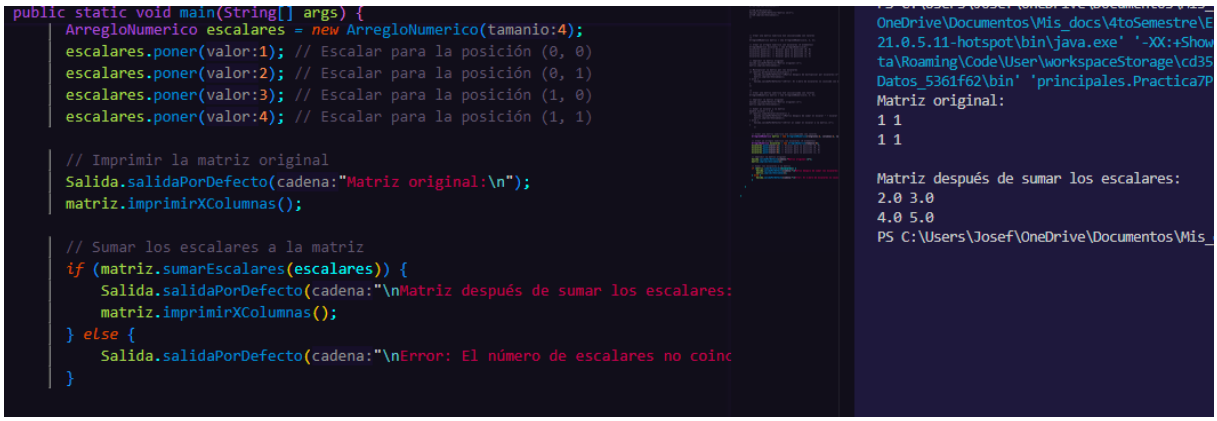


Figure 11: Funcionamiento: sumarEscalares(ArregloNumerico escalares)

2.3.8 boolean multiplicar(Arreglo2DNumerico matriz2)

Multiplicar por otra matriz: boolean multiplicar(Arreglo2DNumerico matriz2). Debe validar las dimensiones y el procedimiento de acuerdo a la teoría de matrices.

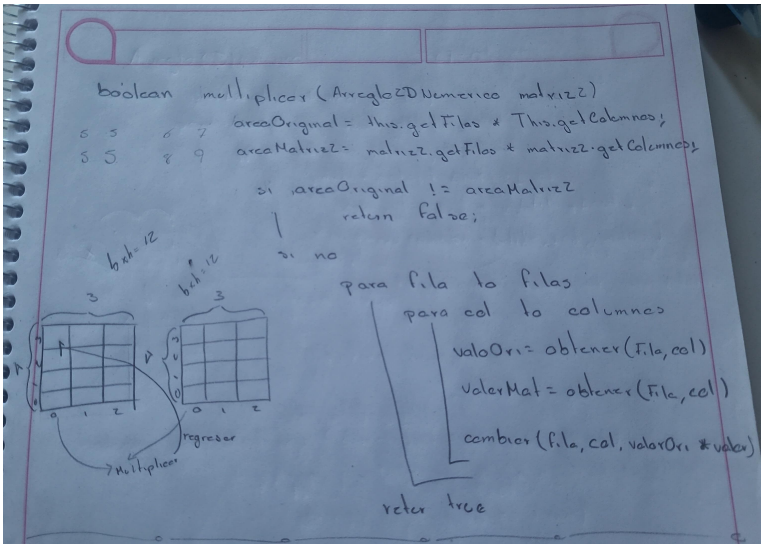


Figure 12: Análisis: multiplicar(Arreglo2DNumerico matriz2)

Explicación

En este caso lo que se hizo fue recuperar la misma posición de ambas matrices, para despue sumar sus valores y regresar el resultado a la misma posición pero en el arreglo original, asi como podemos ver en la figura 7.

```
// Crear la primera matriz numérica 2x2 inicializada con val
Arreglo2DNumerico matriz1 = new Arreglo2DNumerico(renglones:
matriz1.cambiar(fila:0, col:0, valor:1);
matriz1.cambiar(fila:0, col:1, valor:2);
matriz1.cambiar(fila:1, col:0, valor:3);
matriz1.cambiar(fila:1, col:1, valor:4);

// Crear la segunda matriz numérica 2x2 inicializada con val
Arreglo2DNumerico matriz2 = new Arreglo2DNumerico(renglones:
matriz2.cambiar(fila:0, col:0, valor:5);
matriz2.cambiar(fila:0, col:1, valor:6);
matriz2.cambiar(fila:1, col:0, valor:7);
matriz2.cambiar(fila:1, col:1, valor:8);

// Imprimir las matrices originales
Salida.salidaPorDefecto(cadena:"Matriz 1:\n");
matriz1.imprimirXColumnas();

Salida.salidaPorDefecto(cadena:"\nMatriz 2:\n");
matriz2.imprimirXColumnas();

// Multiplicar las matrices
if (matriz1.multiplicar(matriz2)) {
    Salida.salidaPorDefecto(cadena:"\nMatriz 1 después de mu
    matriz1.imprimirXColumnas();
} else {
    Salida.salidaPorDefecto(cadena:"\nError: Las matrices no
}
```

```
deDetailsInExceptionMessages' '-cp' 'C:\Users\
85af\redhat.java\jdt_ws\Estructura-Datos_5361f
Matriz 1:
1 2
3 4

Matriz 2:
5 6
7 8

Matriz 1 después de multiplicar por Matriz 2:
5.0 12.0
21.0 32.0
PS C:\Users\Josef\OneDrive\Documentos\Mis_docs
```

Figure 13: Funcionamiento: multiplicar(Arreglo2DNumerico matriz2)

2.3.9 boolean sumar(Arreglo2DNumerico matriz2)

Sumar con otra matriz: boolean sumar(Arreglo2DNumerico matriz2). Debe validar las dimensiones y el procedimiento de acuerdo a la teoría de matrices.

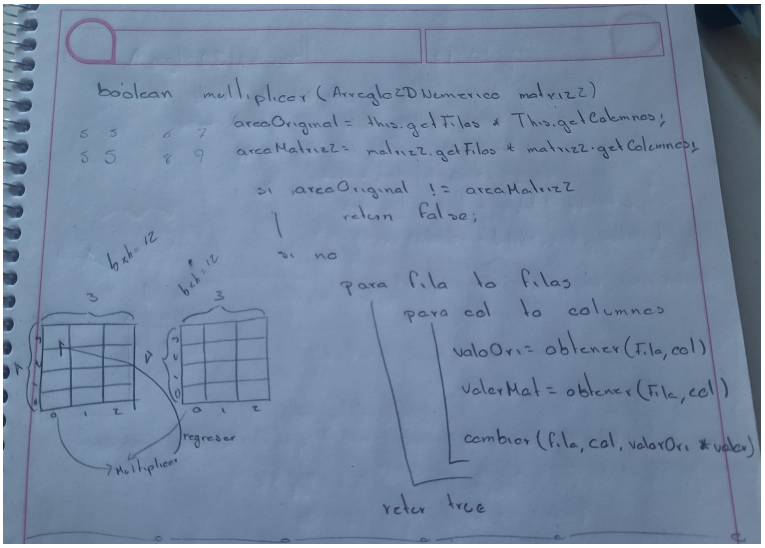
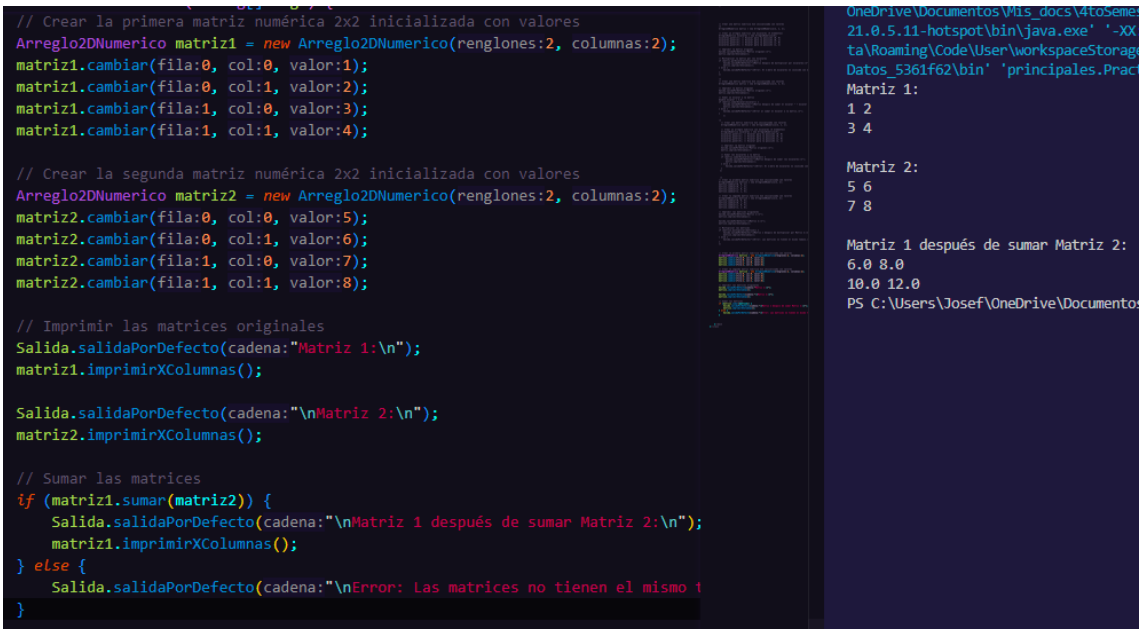


Figure 14: Análisis: sumar(Arreglo2DNumerico matriz2)

Explicación

Aquí lo que se hizo es verificar que las matrices tuvieran el mismo área, ya de no tenerlo regresa false, pero de tener la misma área, se saca los elementos de la misma posición en ambas matrices se suman y se regresa el resultado a la misma posición en la matriz original como podemos ver en la figura 15



```
// Crear la primera matriz numérica 2x2 inicializada con valores
Arreglo2DNumerico matriz1 = new Arreglo2DNumerico(renglones:2, columnas:2);
matriz1.cambiar(fila:0, col:0, valor:1);
matriz1.cambiar(fila:0, col:1, valor:2);
matriz1.cambiar(fila:1, col:0, valor:3);
matriz1.cambiar(fila:1, col:1, valor:4);

// Crear la segunda matriz numérica 2x2 inicializada con valores
Arreglo2DNumerico matriz2 = new Arreglo2DNumerico(renglones:2, columnas:2);
matriz2.cambiar(fila:0, col:0, valor:5);
matriz2.cambiar(fila:0, col:1, valor:6);
matriz2.cambiar(fila:1, col:0, valor:7);
matriz2.cambiar(fila:1, col:1, valor:8);

// Imprimir las matrices originales
Salida.salidaPorDefecto(cadena:"Matriz 1:\n");
matriz1.imprimirXColumnas();

Salida.salidaPorDefecto(cadena:"\nMatriz 2:\n");
matriz2.imprimirXColumnas();

// Sumar las matrices
if (matriz1.sumar(matriz2)) {
    Salida.salidaPorDefecto(cadena:"\nMatriz 1 después de sumar Matriz 2:\n");
    matriz1.imprimirXColumnas();
} else {
    Salida.salidaPorDefecto(cadena:"\nError: Las matrices no tienen el mismo t
}
```

```
OneDrive\Documentos\Mis_docs\4toSeme
21.0.5.11-hotspot\bin\java.exe' '-XX
ta\Roaming\Code\User\workspaceStorag
Datos_5361f62\bin' 'principales.Prac
Matriz 1:
1 2
3 4

Matriz 2:
5 6
7 8

Matriz 1 después de sumar Matriz 2:
6.0 8.0
10.0 12.0
PS C:\Users\Josef\OneDrive\Documento
```

Figure 15: Funcionamiento: sumar(Arreglo2DNumerico matriz2)

2.3.10 boolean aplicarPotencia(Number escalar)

Elevar a una potencia la matriz (elemento por elemento): boolean aplicarPotencia(Number escalar). Aplicar la potencia elemento por elemento.

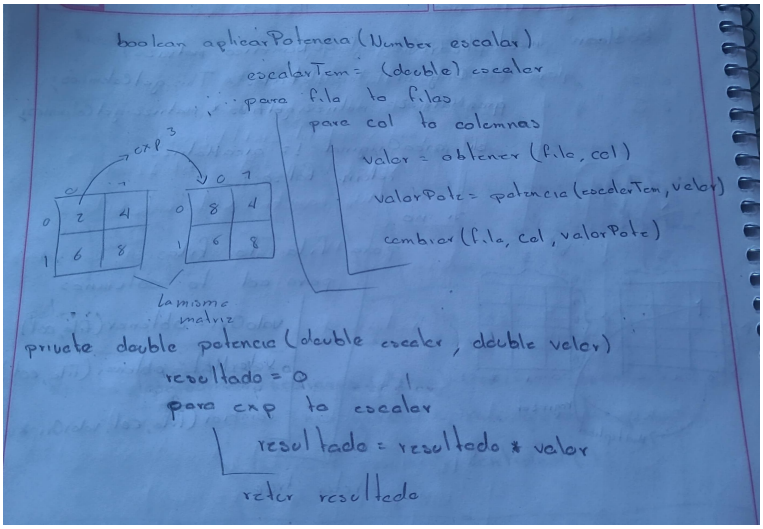


Figure 16: Análisis: aplicarPotencia(Number escalar)

Explicación

Para este caso tuvimos que usar un método auxiliar que lo que hace es recibir un escalar y un valor, lo que área con ello es multiplicar el valor según el valor del escalara a modo de exponencial lo cual nos regresa un double con el valor de esa potencia, para esto sacamos elemento por elemento de la matriz y se lo vamos mandando a este método, para después los resultados ir regresan-dolos a la matriz original en la misma posición, como podemos ver en la figura 17.

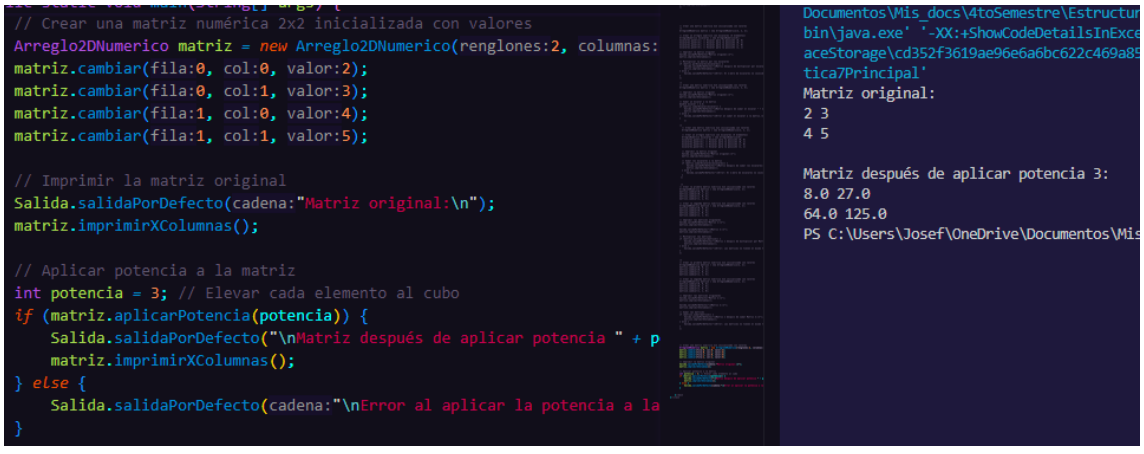


Figure 17: Funcionamiento: aplicarPotencia(Number escalar)

2.3.11 boolean aplicarLogaritmo(TipoLogaritmo tipoLogaritmo)

Aplicar el logaritmo (elemento por elemento) a la matriz: boolean aplicarLogaritmo(TipoLogaritmo tipoLogaritmo). Valide que si existe un valor en cero en la matriz, no se puede aplicar el logaritmo. El proceso es elemento por elemento. Defina un enumerado llamado TipoLogaritmo, que indica que es NATURAL(base e), BASE10 o BASE2. Este tipo será pasado como argumento al método.

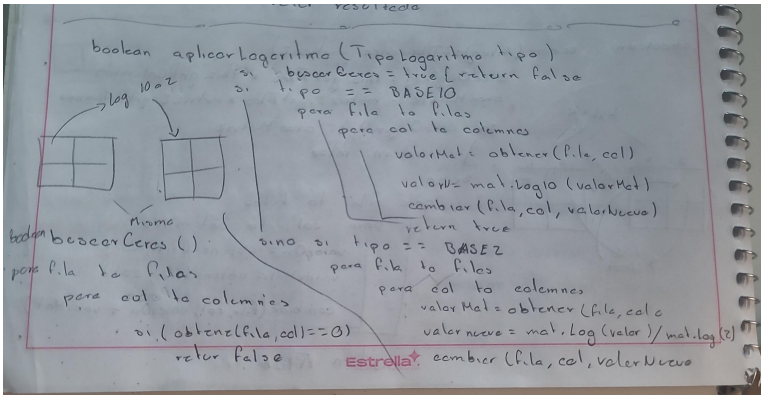


Figure 18: Análisis: aplicarLogaritmo(TipoLogaritmo tipoLogaritmo)

Explicación

Para este caso tuvimos que usar un método auxiliar que lo que hace es revisar sia hay algun cero en la matriz, si hay regresa un true, lo que nos sirve para no aplicar nada sobre la matriz, pero en caso de este regresara un false se empiezan las operaciones, pero sin antes ver cual sabremos gracias con el TipoLogaritmo, ya que este puede ser BASE10 o BASE2, harían lo mismo de sacar cada elemento y aplicarle un logaritmo con la diferencia que uno lo ara en base 10 y otro en base 2 como podemos ver en la figura 19 .

```
matriz.cambiar(fila:0, col:1, valor:4);
matriz.cambiar(fila:1, col:0, valor:8);
matriz.cambiar(fila:1, col:1, valor:16);

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

// Aplicar logaritmo en base 2
if (matriz.aplicarLogartimo(TipoLogaritmo.BASE2)) {
    Salida.salidaPorDefecto(cadena:"\nMatriz después de aplicar logaritmo base
    matriz.imprimirXColumnas();
} else {
    Salida.salidaPorDefecto(cadena:"\nError: No se pudo aplicar el logaritmo ba
}
```

```
Matriz original:
1 10
100 1000

Matriz después de aplicar logaritmo base 10:
0.0 1.0
2.0 3.0

Matriz reiniciada:
2 4
8 16

Matriz después de aplicar logaritmo base 2:
1.0 2.0
3.0 4.0
PS C:\Users\Josef\OneDrive\Documentos\Mis_docs\
```

Figure 19: Funcionamiento: aplicarLogaritmo(TipoLogaritmo tipoLogaritmo)

2.3.12 boolean matrizDiagonal(Number contenido)

Generar una matriz diagonal con el valor proporcionado: boolean matrizDiagonal(Number contenido). al método.

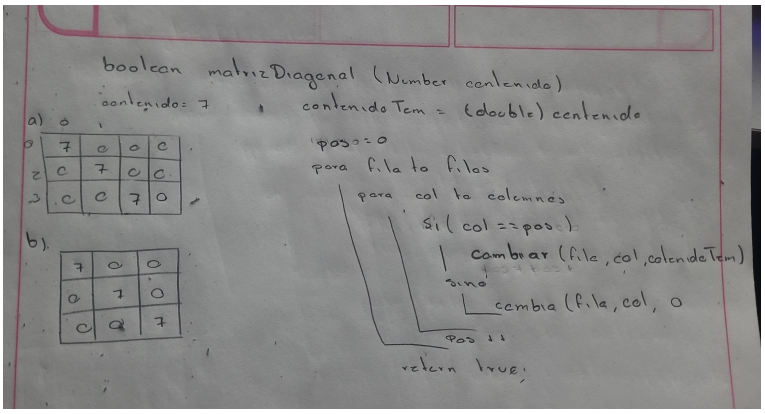


Figure 20: Análisis: matrizDiagonal(Number contenido)

Explicación

Para este caso lo que se hizo fue ver que cada que la columna y fila tuvieran el mismo valor, por ejemplo, (1,1) o (4,4), lo que pasaría es que se escribiría el valor dado, en caso de no ser así, escribiría un cero como podemos ver en la figura 21

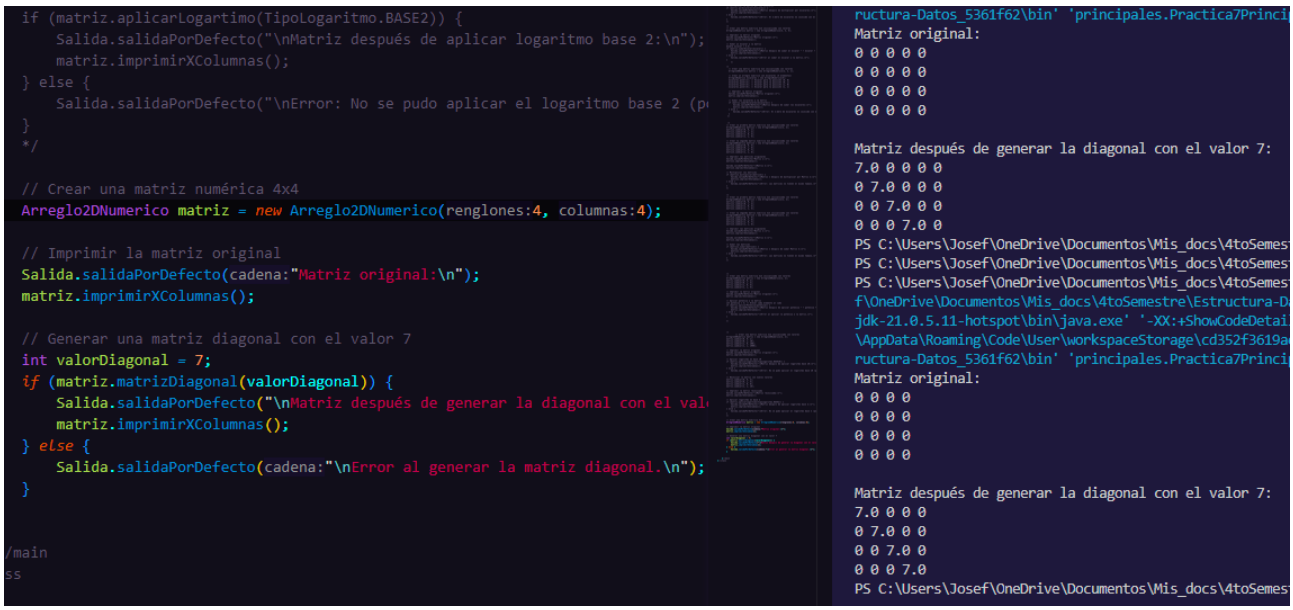


Figure 21: Funcionamiento: matrizDiagonal(Number contenido)

2.3.13 boolean doblarRenglones()

Redimensionar (sumando) una matriz por columnas, a la mitad (si la matriz no tiene un número par de columnas, la del centro debe pasar intacta, solo debe acumular las demás), de tal forma que boolean doblarRenglones():

4	6	4	3
1	2	2	4
5	3	2	1
0	8	7	3

Quedaría así:

10	7
3	6
8	3
8	10

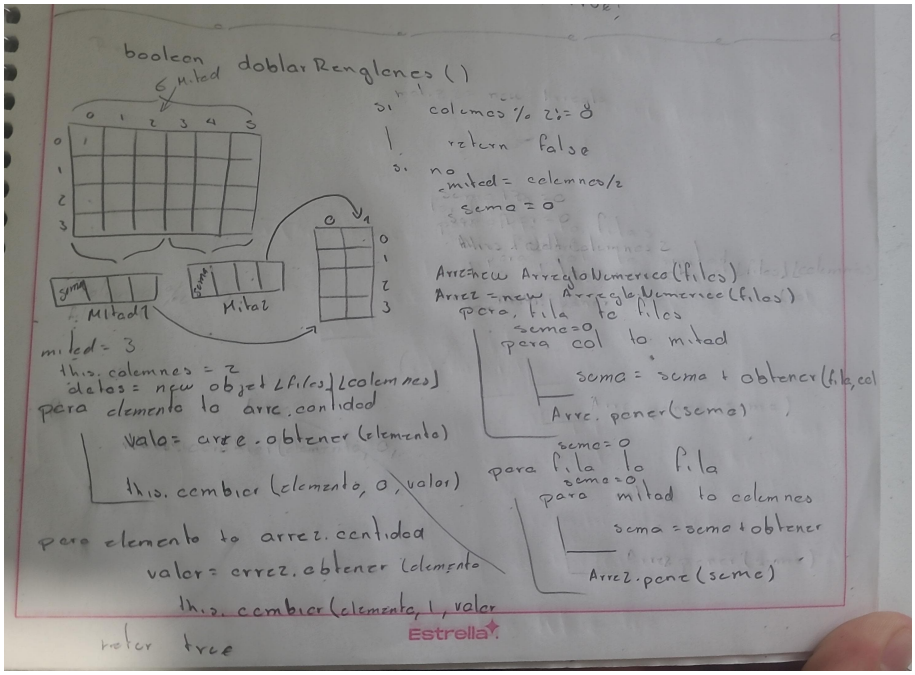


Figure 22: Análisis: doblarRenglones()

Explicación

Lo que se hizo aquí fue que revisamos que que el modulo de la cantidad de columnas con 2 fuera igual a cero, de lo contrario regresa un false, pero de continuar lo que hace es

crear dos arreglos numéricos del tamaño de filas que tiene la matriz, los cuales almacenara la suma de las columnas que van de 0 hasta la mitad y el otro que guarda la suma de columnas de la mitad hasta la cantidad de columnas, después se redefinió el tamaño de la matriz con 2 columnas y la misma cantidad de filas, donde después cada elemento en los arreglos lo pondremos en las columnas de la matriz, como vemos en la figura 23.

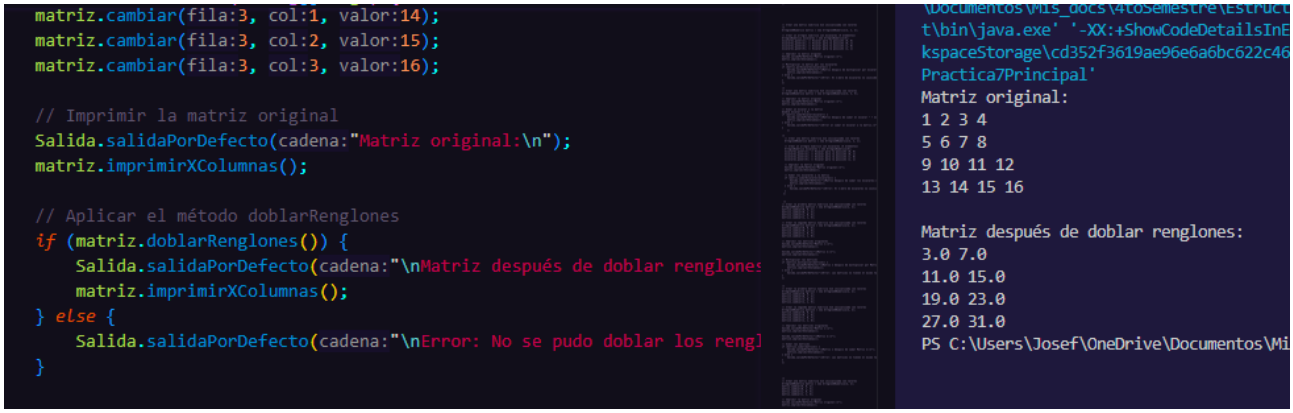


Figure 23: Funcionamiento: doblarRenglones()

2.3.14 boolean doblarColumnas()

. Redimensionar (sumando) una matriz por renglones, a la mitad (si la matriz no tiene un número par de renglones, el del centro debe pasar intacto, solo debe acumular los demás), de tal forma que boolean doblarColumnas():

4	6	4	3
1	2	2	4
5	3	2	1
0	8	7	3

Quedaría así:

5	8	6	7
5	11	9	4

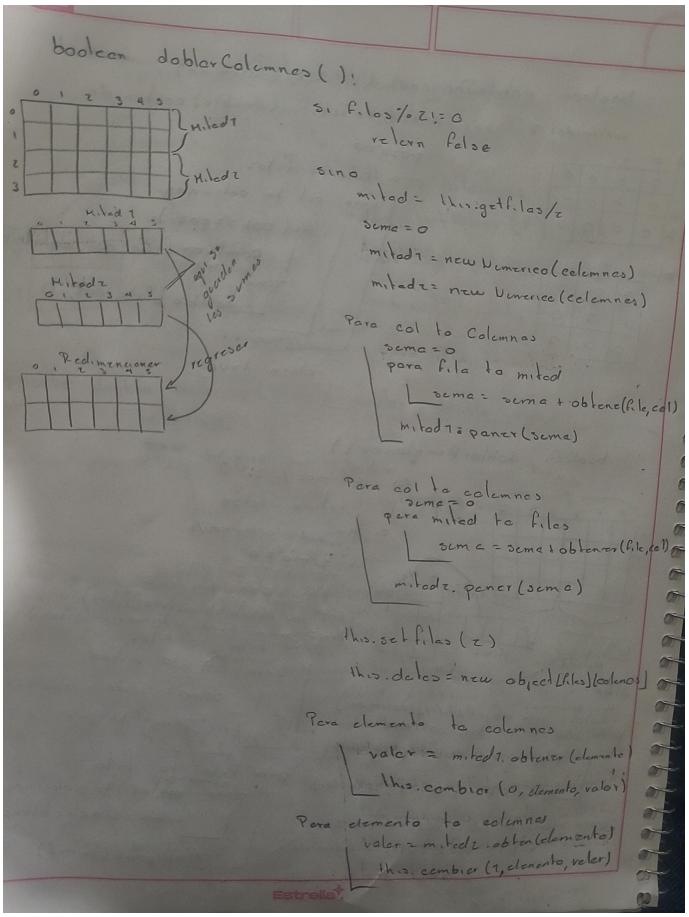


Figure 24: Análisis: doblarColumnas()

Explicación

Lo que se hizo aquí fue que revisamos que que el modulo de la cantidad de filas con 2 fuera igual a cero, de lo contrario regresa un false, pero de continuar lo que hace es crear dos arreglos numéricos del tamaño de columnas que tiene la matriz, los cuales almacenara la suma de las filas que van de 0 hasta la mitad y el otro que guarda la suma de filas de la mitad hasta la cantidad de filas, después se redefinió el tamaño de la matriz con 2 filas y la misma cantidad de columnas, donde después cada elemento en los arreglos lo pondremos en las filas de la matriz, como vemos en la figura 25.

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

// Aplicar el método doblarColumnas
if (matriz.doblarColumnas()) {
    Salida.salidaPorDefecto(cadena:"\nMatriz después de doblar columnas
    matriz.imprimirXColumnas();
} else {
    Salida.salidaPorDefecto(cadena:"\nError: No se pudo doblar las colu
}
```

```
Matriz original:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Matriz después de doblar columnas:
6.0 8.0 10.0 12.0
22.0 24.0 26.0 28.0
PS C:\Users\Josef\OneDrive\Documentos
```

Figure 25: Funcionamiento: doblarColumnas()

3 Código Agregado - UML

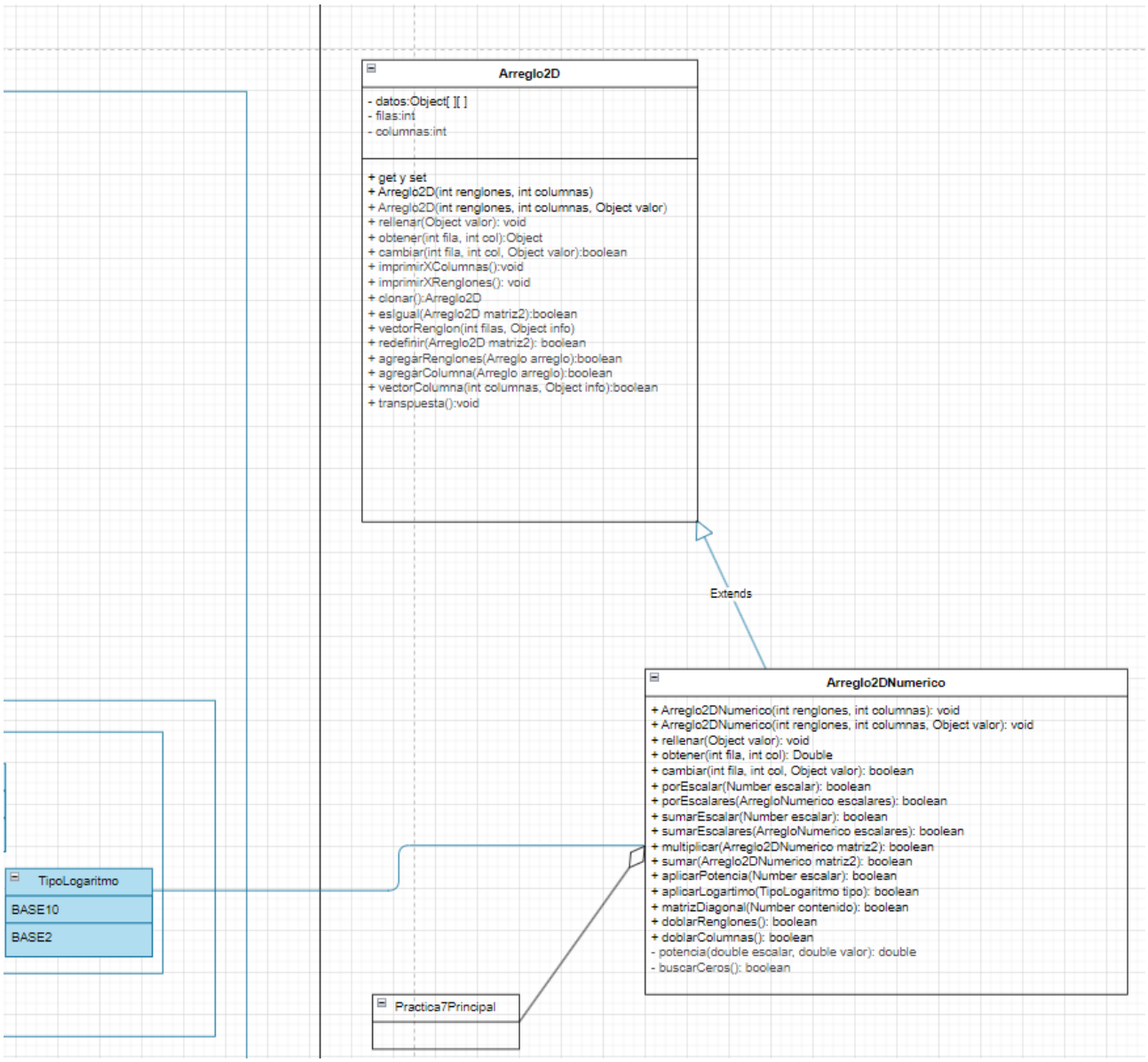


Figure 26: UML

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)	**88%**

Table 1: Evaluación de la práctica

5 Conclusión

Quedo claro el uso de los arreglos2D y arreglos2DNumericos, la verdad estuvo algo complejo de entender pero al final se entendió como se usan

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.