



UNIVERSIDAD AUTÓNOMA DE ZACATECAS

Practica 4

Student:

José Francisco Hurtado Muro

Teacher:

Dr. Aldonso Becerra Sánchez

1 Introducción

"Existe diversidad de usos que se les puede dar a los arreglos. Desde este punto de vista, los arreglos propician que muchos planteamientos puedan tener una solución sencilla si se llevan a cabo con la ayuda de ellos."

Contents

1	Introducción	1
2	Desarrollo	2
2.1	Implementación de Validación de Tipos Numéricos.	2
2.2	boolean porEscalar(Number escalar)	3
2.3	boolean sumarEscalar(Number escalar)	3
2.4	boolean sumar(ArregloNumerico lista2)	4
2.5	boolean multiplicar(ArregloNumerico lista2)	5
2.6	boolean aplicarPotencia(Number escalar)	5
2.7	boolean aplicarPotencia(ArregloNumerico listaEscalares)	6
2.8	double productoEscalar(ArregloNumerico lista2)	7
2.9	double norma()	7
2.10	double normaEuclidiana(ArregloNumerico arreglo2)	8
2.11	boolean sumarListaEstatica(Arreglo listas)	9
2.12	boolean sumarEscalares(ArregloNumerico escalares)	10
2.13	double sumarIndices(ArregloNumerico listaIndices)	11
2.14	[en la clase Arreglo] Arreglo subLista(ArregloNumerico listaIndices)	12
2.15	boolean sonLinealmenteDependientes(Arreglo listaVectores)	13
3	Código Agregado - UML	15
4	Pre-evaluación del Alumno	16
5	Conclusión	16

2 Desarrollo

Clase **ArregloNumerico**. Esta clase tiene como objetivo manejar puros valores numéricos, es decir que hereden de la clase **Number** (utiliza el operador instanceof para facilitar este proceso) cada valor que contendrá. Defina esta clase para que herede de Arreglo. Se le pide que:

2.1 Implementación de Validación de Tipos Numéricos.

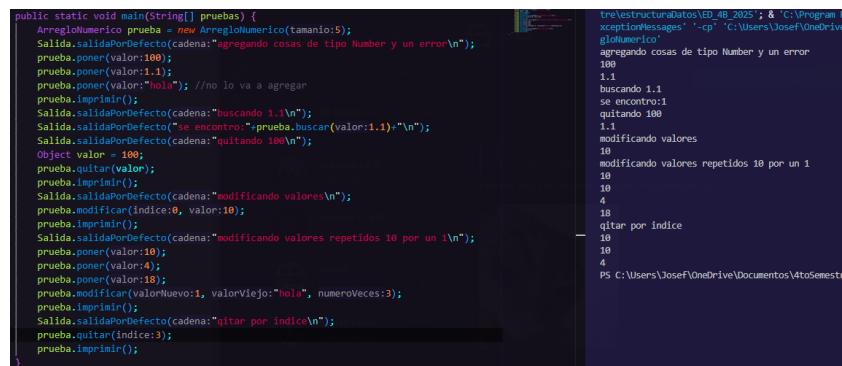
Para ello deberá validar que el contenido que se intenta agregar, eliminar, buscar, modificar y demás operaciones heredadas de Arreglo sean de la clase Number. Para hacer eso tiene que sobre-escribir los métodos necesarios y hacer las invocaciones necesarias a la super clase. Minimice el código reescrito. Si cualquier valor proporcionado no es numérico, debe prohibir la acción del método. Se recomienda convertir todo valor numérico a double, ya que es el máximo en tamaño

```
@Override //debe agregar de tipo Number un numero
public Integer poner(Object valor) {
    if (valor instanceof Number) { //Este metodo hereda de la clase Valores
        return super.poner(valor);
    } else {
        return -1; // Retorna -1 si el valor no es de tipo Number
    }
}
```

Figure 1: Análisis: como usar puras instancias de Number

Explicación

Para que los métodos solo puedan usar Objetos de la clase Number lo que se hizo fue validar que los valores entrantes al método sean instancias de la clase Number y eso se hizo con el **instanceof** y fue usado como podemos ver en la imagen 1, lo cual se aplicó en la mayoría de los métodos heredados donde los parámetros son objetos como: **poner(Object valor)**, **buscar(Object valor)**, **quitar(Object valor)**, **modificar(Object valorNuevo, Object valorViejo, int numeroVeces)**, **modificar(int indice, Object valor)** y **buscarValores(Object valor)**



The screenshot shows a Java code editor and a terminal window. The code in the editor is a test method for an **ArregloNumerico** class. It performs various operations like adding, searching, modifying, and deleting values, and checks for errors when non-numeric values are used. The terminal window shows the execution of the code, including output from `System.out.println` statements and error messages from the `instanceof` operator when it fails to identify a value as a `Number`.

```
public static void main(String[] pruebas) {
    ArregloNumerico prueba = new ArregloNumerico(tamano:5);
    Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
    prueba.poner(valor:100);
    prueba.poner(valor:1.1);
    prueba.poner(valor:'hola'); //no lo va a agregar
    prueba.imprimir();
    Salida.salidaPorDefecto(cadena:"buscando 1.1\n");
    Salida.salidaPorDefecto("se encontró:"+prueba.buscar(valor:1.1)+"\n");
    Salida.salidaPorDefecto(cadena:"quitando 100\n");
    Object valor = 100;
    prueba.quitar(valor);
    prueba.imprimir();
    Salida.salidaPorDefecto(cadena:"modificando valores\n");
    prueba.modificar(indice:0, valor:10);
    prueba.imprimir();
    Salida.salidaPorDefecto(cadena:"modificando valores repetidos 10 por un 1\n");
    prueba.poner(valor:10);
    prueba.poner(valor:4);
    prueba.poner(valor:18);
    prueba.modificar(valorNuevo:1, valorViejo:"hola", numeroVeces:3);
    prueba.imprimir();
    Salida.salidaPorDefecto(cadena:"quitar por indice\n");
    prueba.quitar(indice:3);
    prueba.imprimir();
}
```

Figure 2: Funcionamiento: comprobación de instancias tipo Number

2.2 boolean porEscalar(Number escalar)

multiplicar el escalar dado por cada posición del arreglo numérico. Regrese falso si el arreglo actual está vacío

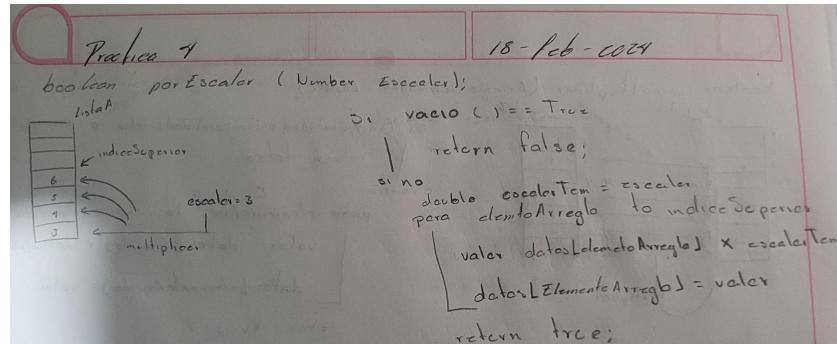


Figure 3: Análisis: método porEscalar

Explicación

Pare este caso lo que se hizo fue revisar que primero no este vacío el arreglo, si lo estuviera regresaría un falso, en caso de si contener algún elemento, entonces el escalar lo transformamos en un double para un mejor manejo de los números, para después iterar sobre cada elemento del arreglo hasta que llegue al límite de elementos en el, y a cada elemento lo multiplicamos por el escalar, una vez teniendo el valor producto regresamos el valor en la posición del valor sacado del arreglo

<pre> ArregloNumerico prueba = new ArregloNumerico(tamano:5); Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n"); prueba.poner(valor:100); prueba.poner(valor:1.1); Salida.salidaPorDefecto(cadena:"usando por escalar con 2\n"); prueba.porEscalar(escalar:2); prueba.imprimir(); </pre>	<pre> CodeDetailsInExceptionMessages -cp C:\US cipales.PrincipalArregloNumerico' agregando cosas de tipo Number y un error usando por escalar con 2 200.0 2.2 PS C:\Users\Josef\OneDrive\Documentos\4toSem </pre>
---	---

Figure 4: Funcionamiento: método porEscalar

2.3 boolean sumarEscalar(Number escalar)

sumar el escalar dado a cada posición del arreglo numérico. Regrese falso si el arreglo actual está vacío

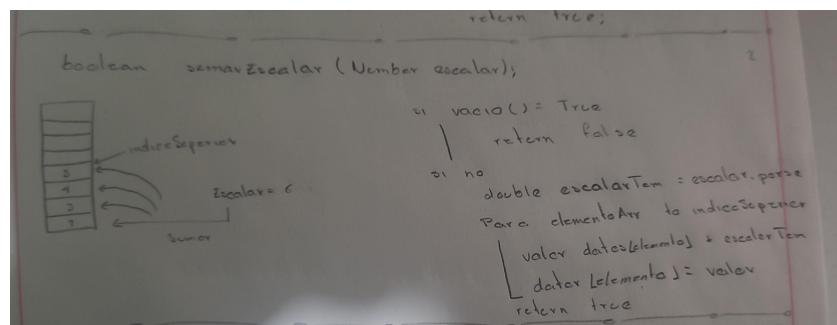


Figure 5: Análisis: método sumarEscalar

Explicación

Este caso no es muy diferente al de la imagen 3 ya que es exactamente lo mismo, verificamos que el arreglo no este vacío, ademas de que el escalar lo pasamos a double, también iteramos en cada elemento del arreglo, para después hacer la operación, que en este caso a cada elemento del arreglo le vamos a sumar el escalar dado

```

ArregloNumerico prueba = new ArregloNumerico(tamano:5);
Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
prueba.poner(valor:100);
prueba.poner(valor:1.1);

Salida.salidaPorDefecto(cadena:"usando sumaEscalar con 2\n");
prueba.sumarEscalar(escalar:2);
prueba.imprimir();
    
```

nMessages -cp "C:\Users\Josef\OneDrive\UX25\bin" 'principales.PrincipalArregloNumerico
agregando cosas de tipo Number y un error
usando sumaEscalar con 2
102.0
3.1
PS C:\Users\Josef\OneDrive\Documentos\4toSem

Figure 6: Funcionamiento: método sumarEscalar

2.4 boolean sumar(ArregloNumerico lista2)

sumar la posición 1 del arreglo actual con la posición 1 de arreglo2, y así sucesivamente. Note que los valores pueden ser negativos, en tal caso debe seguir funcionando este proceso. Valide dimensiones

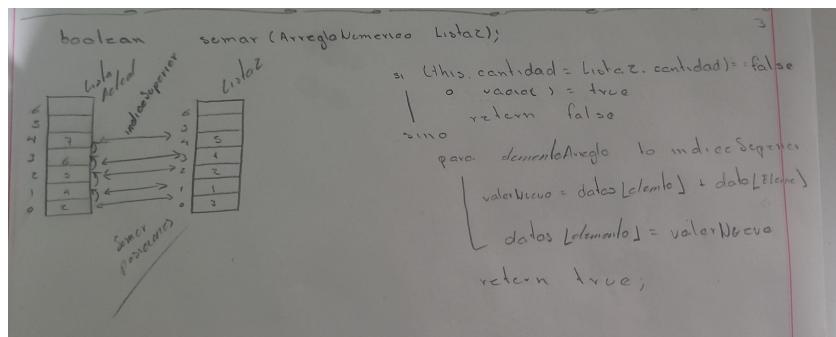


Figure 7: Análisis: método sumar

Explicación

Aquí lo que se hizo fue recibir una lista del tipo ArregloNumerico, el cual debemos sumar posición con posición, por lo cual para esto revisaremos que ambos arreglos tengan la misma cantidad de elementos ya que si existiera un elemento de menos en de los arreglos no lo podríamos sumar con un nulo, pero en caso de pasar este primera parte, lo que hará es ir sobre cada elemento de los dos arreglos sumando el valor de la posición con la posición del arreglo2 y lo regresaría de nuevo a la posición del arreglo actual

```

ArregloNumerico prueba = new ArregloNumerico(tamano:5);
Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
prueba.poner(valor:5);
prueba.poner(valor:1.1);

ArregloNumerico lista2 = new ArregloNumerico(tamano:4);
lista2.poner(valor:3);
lista2.poner(valor:2.1);

Salida.salidaPorDefecto(cadena:"usando sumar com lista 2\n");
prueba.sumar(lista2);
prueba.imprimir();
    
```

nMessages -cp "C:\Users\Josef\OneDrive\UX25\bin" 'principales.PrincipalArregloNumerico
agregando cosas de tipo Number y un error
usando sumar com lista 2
8.0
3.2
PS C:\Users\Josef\OneDrive\Documentos\4toSem

Figure 8: Funcionamiento: método sumar

2.5 boolean multiplicar(ArregloNumerico lista2)

que haga producto de posición 1 del arreglo actual por posición 1 de arreglo2, y así sucesivamente. Regrese falso si el arreglo actual está vacío

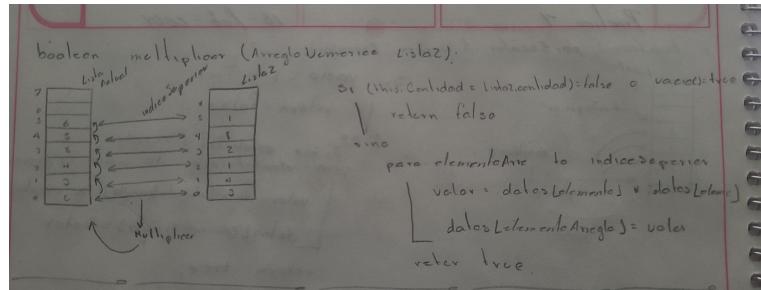


Figure 9: Análisis: método multiplicar

Explicación

Bueno en este caso es similar al de la imagen 7 ya que hace exactamente lo mismo, revisa que tenga la misma cantidad de elementos en ambos arreglos, si no regresa false, en caso de pasar de aquí itera sobre ambos arreglos y multiplica la posición del arreglo actual con el de la lista2 y lo regresa como el nuevo valor de la posición del arreglo actual

```
public static void main(String[] args) {
    ArregloNumerico prueba = new ArregloNumerico(tamano:5);
    Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
    prueba.poner(valor:5);
    prueba.poner(valor:1.1);
    prueba.poner(-2);

    ArregloNumerico lista2 = new ArregloNumerico(tamano:4);
    lista2.poner(valor:3);
    lista2.poner(valor:2.1);
    lista2.poner(-1);

    Salida.salidaPorDefecto(cadena:"usando multiplicar con lista 2\n");
    prueba.multiplicar(lista2);
    prueba.imprimir();
}
```

Figure 10: Funcionamiento: método multiplicar

2.6 boolean aplicarPotencia(Number escalar)

que haga la operación de potencia de cada elemento del arreglo (base) por el exponente pasado como escalar. Regrese falso si el arreglo actual está vacío

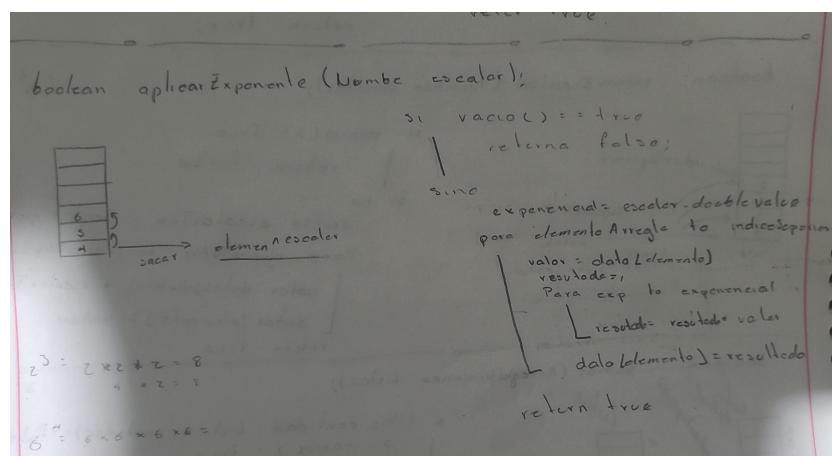


Figure 11: Análisis: método aplicarPotencia

Explicación

Bueno en este caso lo que se hizo fue ver que el arreglo no estuviera vacío, si lo estuviera regresaría un false, pero si llega a pasar a la siguiente parte, lo que pasaría sería que iteraría sobre cada elemento del arreglo aplicándole a cada elemento la operación de multiplicarlo por su mismo valor hasta que el exponente deseado y el resultado se guarda en la posición del elemento del arreglo actual

```
Salida.salidaPorDefecto(cadena:"usando aplicando exponecial 2\n");
lista2.aplicarExponencial(escalar:2);
lista2.imprimir();
```

agregando cosas de tipo Number y un error
usando aplicando exponecial 2
9.0
4.0
PS C:\Users\Josef\OneDrive\Documentos\4toS

Figure 12: Funcionamiento: método aplicarPotencia

2.7 boolean aplicarPotencia(AregloNumerico listaEscalares)

que haga la operación de potencia de cada elemento del arreglo (base) por el exponente pasado como arreglo, posición por posición. Valide dimensiones.

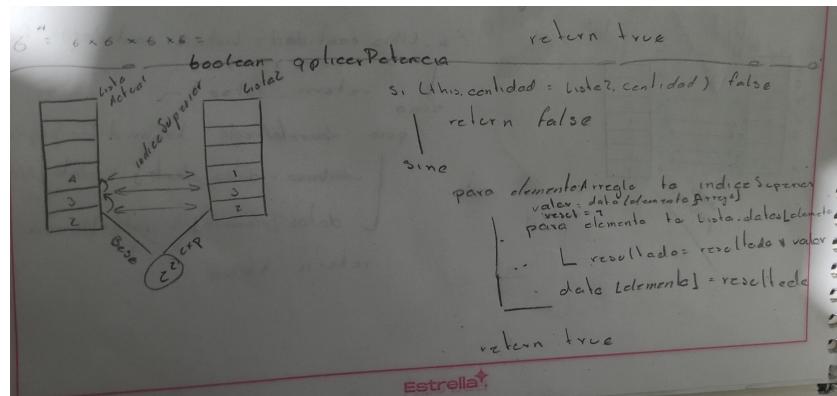


Figure 13: Análisis: método aplicarPotencia con lista

Explicación

Aquí hacemos lo mismo que en la imagen 11 con la diferencia de aquí recibimos una lista de tipo AregloNumerico el cual contiene los exponentiales, y al momento de aplicar la operación, el límite es sacado del mismo nivel que el elemento de la lista actual

```
public static void main(String[] pruebas) {
    ArregloNumerico prueba = new ArregloNumerico(tamano:5);
    Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
    prueba.poner(valor:6);
    prueba.poner(valor:3);
    prueba.poner(valor:9);

    ArregloNumerico lista2 = new ArregloNumerico(tamano:4);
    lista2.poner(valor:5);
    lista2.poner(valor:3);
    lista2.poner(valor:4);

    Salida.salidaPorDefecto(cadena:"usando aplicando exponecial con lista\n");
    prueba.aplicarPotencia(lista2);
    prueba.imprimir();
```

oSemestre\estructuralDatos\ED_4B_2025'; &
 deDetailsInExceptionMessages' '-cp 'C:\Users\PrincipialArregloNumerico'
 agregando cosas de tipo Number y un error
 usando aplicando exponecial con lista
 776.0
 27.0
 6561.0
 PS C:\Users\Josef\OneDrive\Documentos\4toS

Figure 14: Funcionamiento: método aplicarPotencia con lista

2.8 double productoEscalar(ArregloNumerico lista2)

por ejemplo considere los vectores matemáticos (arreglos) siguientes de dimensión 3: a, b, c • x, y, zT = a•x + b•y + c•z. Donde la T es la transpuesta del vector y • es la multiplicación elemento por elemento

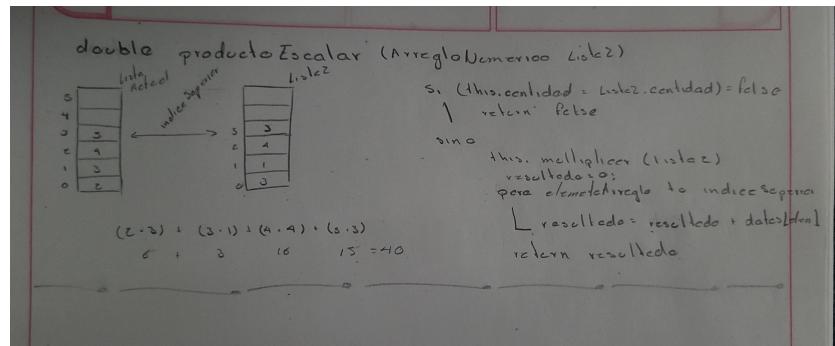


Figure 15: Análisis: método productoEscalar

Explicación

En este caso, lo que se hizo fue revisar que ambos arreglos tengan la misma cantidad de elementos, una vez validada que la cantidad de elemento sea la misma, procedemos a aplicar el método ya elaborado **multiplicar**, que multiplica por indice los elementos de dos listas, y para obtener el producto escalar sumamos todos los resultados de multiplicación

```

ArregloNumerico prueba = new ArregloNumerico(tamano:5);
Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error");
prueba.poner(valor:2);
prueba.poner(valor:3); booleana sumar(ArregloNumerico lista2); sumar
prueba.poner(valor:4); con la posición 1 de arreglo2, y así sucesivamente
prueba.poner(valor:5); ser negativos, en tal caso debe seguir función
ArregloNumerico lista2 = new ArregloNumerico(tamano:4);
lista2.poner(valor:3); booleana multiplicar(ArregloNumerico lista2); que haga producto de posición 1
lista2.poner(valor:1); del arreglo actual por posición 1 de arreglo2, y así sucesivamente. Regrese falso
lista2.poner(valor:4); si el arreglo actual está vacío.
lista2.poner(valor:3); booleana aplicarPotencia(Number escalar); que haga la operación de potencia de
Salida.salidaPorDefecto(cadena:"sacando producto escalar \n"); el exponente pasado como escalar. Regrese
Salida.salidaPorDefecto(""+prueba.productoEscalar(lista2));
    
```

Figure 16: Funcionamiento: método productoEscalar

2.9 double norma()

saca la magnitud / módulo / norma L2 del vector (arreglo); por ejemplo considere el siguiente vector numérico de dimensión 3: V= a, b, c, $\|V\| = \sqrt{a^2 + b^2 + c^2}$. Donde \sqrt es la raíz cuadrada. la T es la transpuesta del vector y • es la multiplicación elemento por elemento

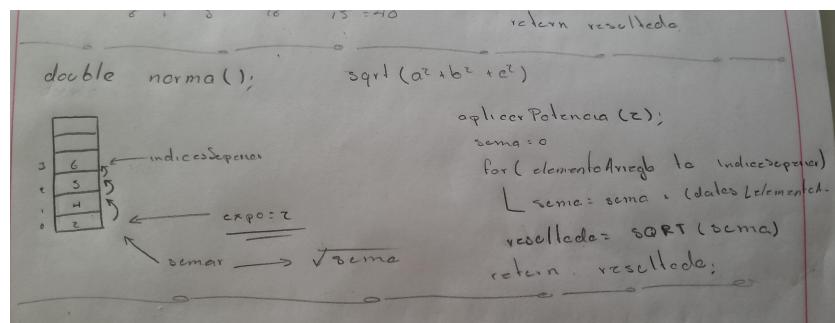


Figure 17: Análisis: método norma

Explicación

Para este caso lo que se hizo primero fue aplicar el método ya elaborado de **aplicarPotencia** con el parámetro 2, para después sumar todos los elementos del arreglo, para poder sacarles raíz cuadrada, que de manera matemática se ve así $\|\mathbf{V}\| = \sqrt{a^2 + b^2 + c^2}$

```

public static void main(String[] pruebas) {
    ArregloNumerico prueba = new ArregloNumerico(tamano:5);
    Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error");
    prueba.poner(valor:2);
    prueba.poner(valor:3);
    prueba.poner(valor:4);
    prueba.poner(valor:5);

    ArregloNumerico lista2 = new ArregloNumerico(tamano:4);
    lista2.poner(valor:3);
    lista2.poner(valor:1);
    lista2.poner(valor:4);
    lista2.poner(valor:3);

    Salida.salidaPorDefecto(cadena:"calculando la norma \n");
    Salida.salidaPorDefecto(""+prueba.norma());
}
  
```

ArregloNumerico' agregando cosas de tipo Number y un error calculando la norma 7.3484692283495345 PS C:\Users\Josef\OneDrive\Documentos\4toSem

Figure 18: Funcionamiento: método norma

2.10 double normaEuclidian(a ArregloNumerico arreglo2)

debe calcular la norma euclidiana del vector numérico AB (arreglos n dimensionales). Este cálculo está dado por:

$$\|AB\| = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \dots + (b_n - a_n)^2}$$

donde A=a1, a2, a3, ..., an y B=b1, b2, b3, ..., bn.

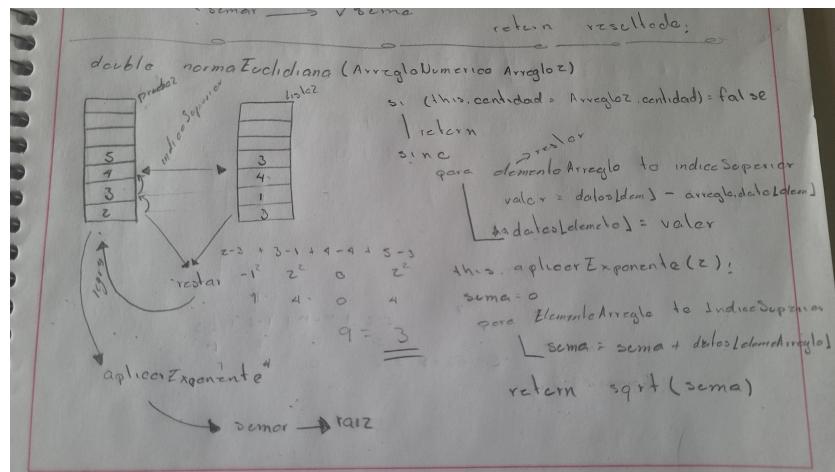


Figure 19: Análisis: método normaEuclidianas

Explicación

Para este caso lo que se hizo primero fue aplicar el método ya elaborado de **aplicarPotencia** con el parámetro 2, para después sumar todos los elementos del arreglo, para poder sacarles raíz cuadrada, que de manera matemática se ve así $\|V\| = \sqrt{a^2 + b^2 + c^2}$

```

public static void main(String[] pruebas) {
    ArregloNumerico prueba = new ArregloNumerico(tamano:5);
    Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
    prueba.poner(valor:2); // double productoscalar ArregloNumerico lista2); por
    prueba.poner(valor:3); // vectores matemáticos (arreglos) siguientes de dimensión
    prueba.poner(valor:4); // = a*x + b*y + c*z. Donde la {{}}^T es la transpuesta del vector
    prueba.poner(valor:5); // elemento por elemento.
    ArregloNumerico lista2 = new ArregloNumerico(tamano:4); magnitud / módulo / norma L2 del vector (arreglo); por
    lista2.poner(valor:3); // ejemplo considere el siguiente vector numérico de dimensión 3: V= {a, b, c}.
    lista2.poner(valor:1); // ||V||= sqrt(a^2 + b^2 + c^2). Donde sqrt es la raíz cuadrada.
    lista2.poner(valor:4); // double normaEuclidianas(ArregloNumerico arreglo2); debe calcular la norma
    lista2.poner(valor:3); // elemento por elemento de los arreglos n dimensionales. Este cálculo está
    Salida.salidaPorDefecto(cadena:"calculando la norma euclidiana \n");
    Salida.salidaPorDefecto(""+lista2.normaEuclidianas(prueba));
}
    
```

Figure 20: Funcionamiento: método normaEuclidianas

2.11 boolean sumarListaEstatica(Arreglo listas)

debe sumar de uno por uno (como el método de arriba pero ahora son varias listas) un conjunto de arreglos de tipo ArregloNumerico almacenados en la variable listas al arreglo actual. Cada posición de listas guarda un arreglo

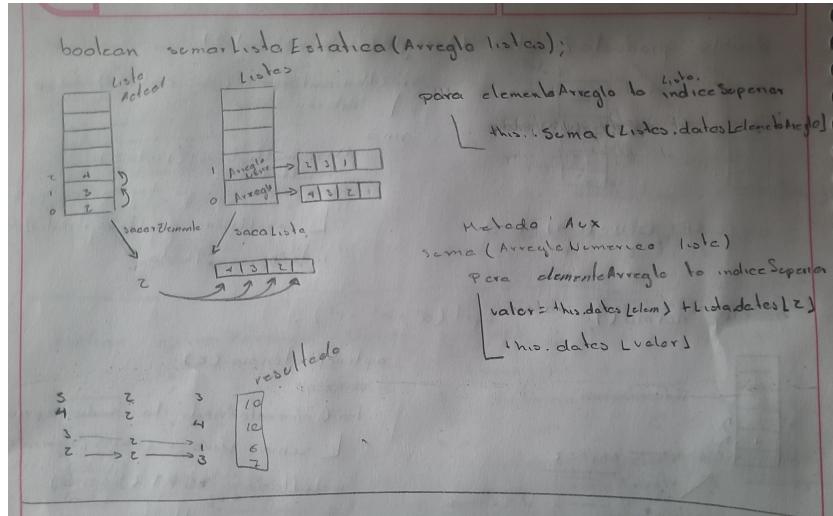


Figure 21: Análisis: método sumarListaEstatica

Explicación

Para este caso lo que se hizo fue elaborar un método auxiliar para facilitar el método **sumarListaEstatica**, este método auxiliar tiene como objetivo recibir un `ArregloNumerico` el cual sumara posición con posición de ambos arreglos, actual y el recibido, y el resultado se va guardando en el Arreglo actual en la misma posición de donde se saco el valor, para el método principal lo que se hizo hacer una iteración en los elementos de la lista de arreglos, y ir aplicando el método auxiliar

```

Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
prueba.poner(valor:2);
prueba.poner(valor:3);
prueba.poner(valor:4);
prueba.poner(valor:5);

ArregloNumerico lista = new ArregloNumerico(tamano:5);
lista.poner(valor:2);
lista.poner(valor:2);
lista.poner(valor:2);
lista.poner(valor:2);
ArregloNumerico lista2 = new ArregloNumerico(tamano:4);
lista2.poner(valor:3);
lista2.poner(valor:1);
lista2.poner(valor:4);
lista2.poner(valor:3); //Este caso lo que se hizo primero fue aplicar el método ya elaborado de <control+aplicar> para el primer parámetro de lista2, para después sumar todos los elementos del arreglo, para poder sacarlos en la lista actual, para esto se aplica el método ya elaborado de <control+aplicar> para el segundo parámetro de lista2, para sacar los resultados de la lista2 y ponerlos en la lista actual.

Arreglo listas = new Arreglo(tamano:3); //arreglo de arreglosNumericos
listas.poner(lista);
listas.poner(lista);
listas.poner(lista2);

Salida.salidaPorDefecto(cadena:"probando sumar lista estatica \n");
prueba.sumarListaEstatica(listas);
prueba.imprimir();

```

Output terminal:

```

eDrive\Documentos\4toSemestre\estructuraDatos
.5.11-hotspot\bin\java.exe' '-XX:+ShowCodeDetails
umentos\4toSemestre\estructuraDatos\ED_4B_202
agregando cosas de tipo Number y un error
probando sumar lista estatica
7.0
6.0
10.0
10.0
PS C:\Users\Josef\OneDrive\Documentos\4toSeme

```

Figure 22: Funcionamiento: método sumarListaEstatica

2.12 boolean sumarEscalares(ArregloNumerico escalares)

debe sumar de uno por uno un conjunto de escalares almacenados en la variable escalares al arreglo actual. Escalares es un arreglo que guarda en cada posición un escalar.

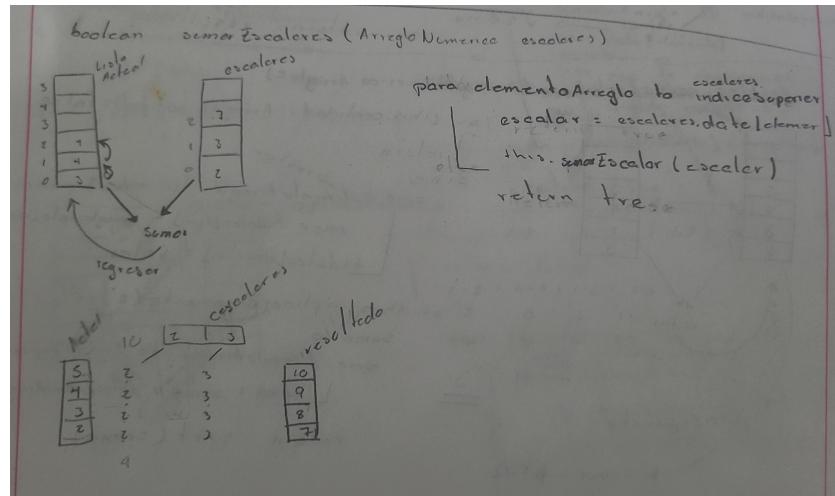


Figure 23: Análisis: método sumarEscalares

Explicación

Para este método lo que se hizo fue recorrer el ArregloNumerico enviando he ir sacando cada elemento, para después mandárselo al método ya echo **sumarEscalar**, quien se encarga de aplicarle la suma de escalares al arreglo actual

```

ArregloNumerico lista = new ArregloNumerico(tamano:5); double norma(); saca la magnitud / modulo
lista.poner(valor:2);
lista.poner(valor:3);

ArregloNumerico lista2 = new ArregloNumerico(tamano:4); double normaEuclidiana(ArregloNumerico ar);
lista2.poner(valor:3);
lista2.poner(valor:1);
lista2.poner(valor:4);
lista2.poner(valor:3);

Salida.salidaPorDefecto(cadena:"probando sumarEscalares \n");
prueba.sumarEscalares(lista);
prueba.sumarIndices(listaIndices);
prueba.imprimir();
    
```

agregando cosas de tipo Number y un error probando sumarEscalares
 7.0
 8.0
 9.0
 10.0
 PS C:\Users\Josef\OneDrive\Documentos\4toS

Figure 24: Funcionamiento: método sumarEscalares

2.13 double sumarIndices(ArregloNumerico listaIndices)

Debe sumar, del arreglo actual, las posiciones de él que indica el arreglo llamado listaIndices, el cual almacena las posiciones que se deben tomar del arreglo actual para hacer la suma.

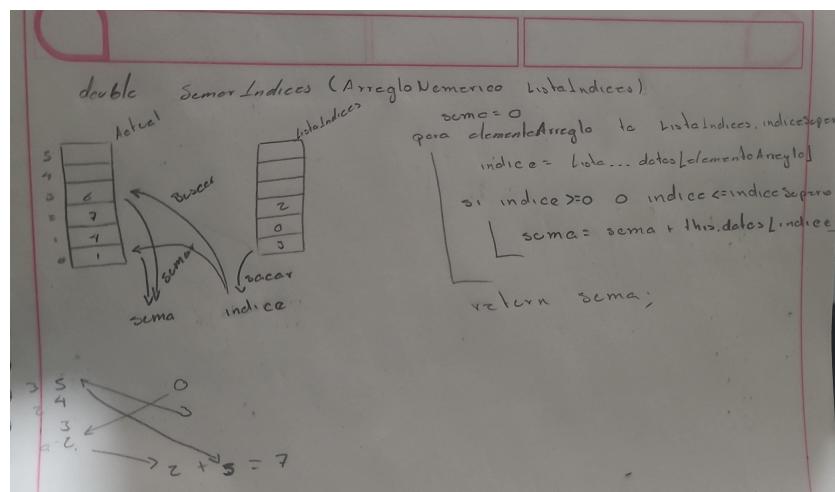


Figure 25: Análisis: método sumarEscalares

Explicación

Aquí lo que se hizo fue iterar sobre la listaIndices, y ir recuperando sus valores, después validamos que ese índice estuviera en dentro del rango de elementos en el arreglo actual, para después extraer el valor de ese índice y irlo sumando

```

ArregloNumerico prueba = new ArregloNumerico(tamano:5);
Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
prueba.poner(valor:2);
prueba.poner(valor:3);
prueba.poner(valor:4);
prueba.poner(valor:5);

ArregloNumerico lista = new ArregloNumerico(tamano:5);
lista.poner(valor:0);
lista.poner(valor:3);

Salida.salidaPorDefecto(cadena:"probando sumaEscalares \n");
Salida.salidaPorDefecto(""+prueba.sumarIndices(lista));

```

.11-hotspot\bin\java.exe` `XX:+ShowCodeDe
ntos\4toSemestre\estructuraDatos\ED_4B_202
agregando cosas de tipo Number y un error
probando sumaEscalares
7.0
PS C:\Users\Josef\OneDrive\Documentos\4toS
rreglo2); debe calcular la norma
dimensionales). Este cálculo se d
dado por:

$$\|V\| = \sqrt{a^2 + b^2 + c^2}$$

double normaEuclidianas(ArregloNumerico
euclidianas del vector numérico AB (arreglos
dimensionales). Este cálculo se d
dado por:

$$\|AB\| = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \dots + (b_n - a_n)^2}$$

double norma(A:ArregloNumerico, B:ArregloNumerico); donde A=[a₁, a₂, a₃, ..., a_n] y B=[b₁, b₂, b₃, ..., b_n]

Figure 26: Funcionamiento: método sumarEscalares

2.14 [en la clase Arreglo] Arreglo subLista(ArregloNumerico listaIndices)

Debe regresar un arreglo contenido los elementos del arreglo actual que se obtienen del arreglo de índices “listaIndices”, el cual contiene las posiciones de los índices de donde se obtendrán los datos a retornar. Significa que se deberán hacer dos métodos, el correspondiente en la super clase (Arreglo), para datos generales y el correspondiente a la clase ArregloNumerico, correspondiente solo a números

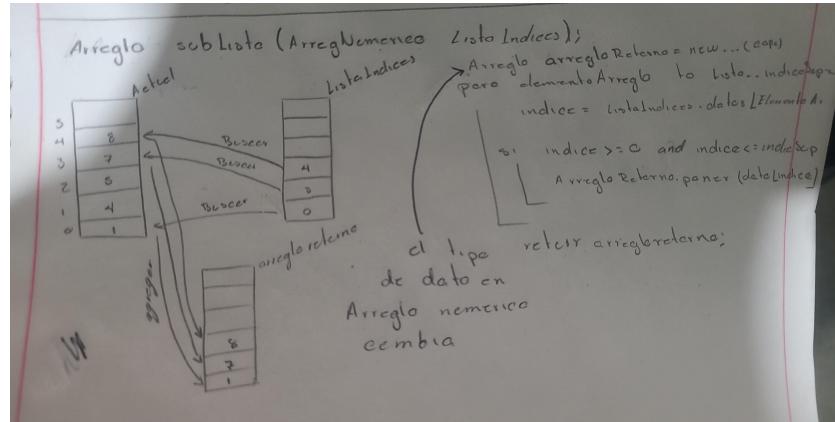


Figure 27: Análisis: método subLista

Explicación

Bueno en este caso se hizo el método en la clase **Arreglo**, el cual consiste en crear un arreglo temporal, para después iterar sobre la lista indice obteniendo los índices de las posiciones en el arreglo actual, las cuales se validan y una vez validadas se sacan los datos que correspondan en ese índice y se agrega al arreglo temporal para después regresarlo. en la clase **ArregloNumerico** no se hizo gran cambio ya que el tipo de dato no afecta mucho ademas de que es la misma clase que se encarga de meter valores validos, por esto mismo el unico cambio fue el tipo de arrgo temporal, que en este caso se quedo como ArregloNumerico y no como Arreglo

```

public static void main(String[] pruebas) {
    Salida.salidaPorDefecto(cadena:"agregando cosas de tipo Number y un error\n");
    prueba.poner(valor:2);
    prueba.poner(valor:3);
    prueba.poner(valor:4);
    prueba.poner(valor:5);

    ArregloNumerico lista = new ArregloNumerico(tamano:5);
    lista.poner(valor:0);
    lista.poner(valor:3);

    Salida.salidaPorDefecto(cadena:"probando subLista \n");
    Arreglo arre = prueba.subLista(lista);
    arre.imprimir();
}
    
```

posición de listas guarda un arreglo.
 boolean sumarEscalares(ArregloNumerico lista);
 un conjunto de escalares almacenados en la lista.
 Escalares es un arreglo que guarda en cada posición un escalar.
 double sumarIndices(ArregloNumerico listaIndices); Debe sumar, del arreglo
 actual, las posiciones de él que indica el arreglo llamado listaIndices, el cual
 almacena las posiciones que se deben tomar del arreglo actual para hacer la suma.
 double subList(ArregloNumerico lista, ArregloNumerico listaIndices); Debe

Figure 28: Funcionamiento: método subLista

2.15 boolean sonLinealmenteDependientes(Arreglo listaVectores)

Teniendo en cuenta la siguiente definición: sean v_1, v_2, \dots, v_n n vectores de un espacio vectorial V . Se dice que dichos vectores son **linealmente dependientes** (L.D.) si existen n escalares k_1, k_2, \dots, k_n , no todos nulos, tales que

$$k_1v_1 + k_2v_2 + \cdots + k_nv_n = 0.$$

Haga un método boolean `sonLinealmenteDependientes(Arreglo listaVectores)` que determine si el conjunto de arreglos pasado como argumentos (es un arreglo de arreglos numéricos) (representa V) son linealmente dependientes. De tal forma que el arreglo actual es el conjunto de n escalares K .

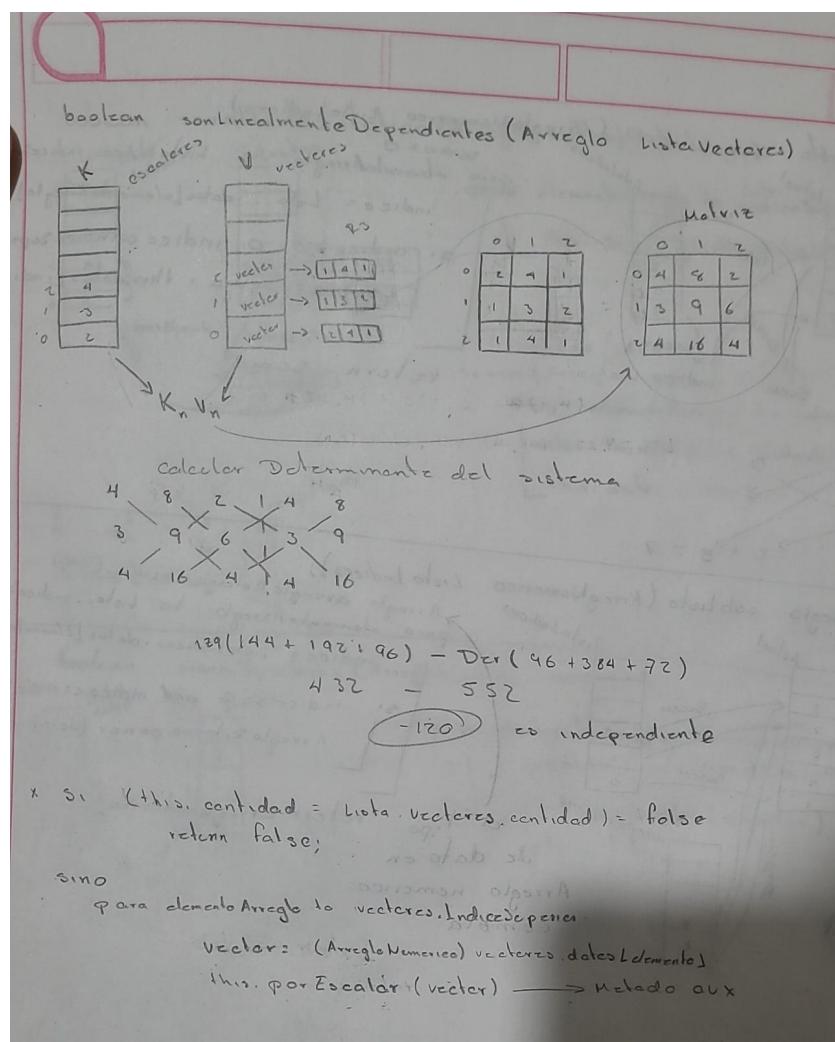


Figure 29: Análisis: método sonLinealmenteDependientes

Explicación

Bueno lo que se intento hacer es que para cada elemento de listaVectores, se le aplique un escalar, después esto tendríamos que resolverlo por determinantes, donde si el determinante del sistema es diferente a cero, significa que no es dependiente

3 Código Agregado - UML

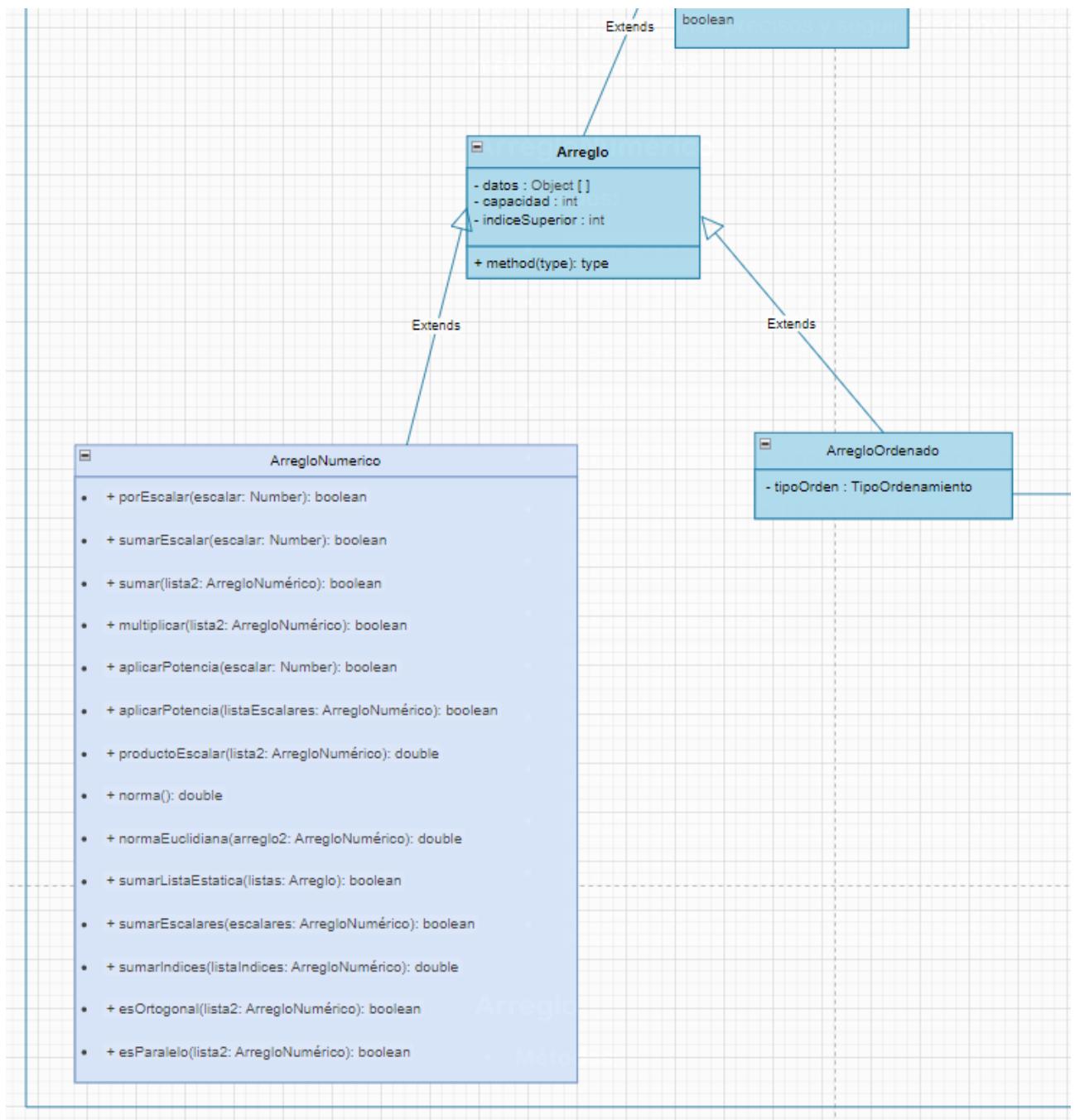


Figure 30: UML

4 Pre-evaluación del Alumno

Práctica NO: 4

José Francisco Hurtado Muro

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)	90%

Table 1: Evaluación de la práctica

5 Conclusión

se vio que si son útiles los arreglos numérico en especial para aplicación de cosas matemáticas como el álgebra lineal para combinaciones lineales