



UNIVERSIDAD AUTÓNOMA DE ZACATECAS

Practica 3

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	Análisis 1	2
2.2	Análisis 2	3
2.3	Análisis 3	3
2.4	Análisis 4	4
2.5	Análisis 5	5
2.6	Análisis 6	6
2.7	Análisis 7	6
2.8	Análisis 8	7
2.9	Análisis 9	8
2.10	Análisis 10	9
2.11	Análisis 11	10
2.12	Análisis 12	11
2.13	Análisis 13	12
2.14	Análisis 14	13
3	Código Agregado - UML	16
4	Pre-evaluación del Alumno	17
5	Conclusión	17

2 Desarrollo

Actividad 1

Defina el TDA llamado “ArregloOrdenado” (que herede de Arreglo, tal cual se muestra en el diagrama de clases proporcionado en sesiones anteriores), el cual debe tener atributos y métodos (los cuales deberán sobre-escribirse según sea el caso; utilice el estereotipo @Override antes del método para garantizar la correcta verificación de sobreescritura en la herencia). Recuerde que el orden en números no es lo mismo que el orden en cadenas y otros tipos de objetos; ya que debe personalizar este mecanismo según sea el tipo de contenido. Para poder hacer la práctica se ocupa un módulo comparador de objetos de cualquier tipo (pero los dos objetos deben ser del mismo tipo).

Nota: no utilice ningún método de ordenamiento formal por lotes.

Se pide que realice los siguientes métodos dentro de esta clase (deben estar enfocados para datos estrictamente ordenados, lo cual no es igual para datos desordenados):

2.1 Análisis 1

public ArregloOrden(int maximo, TipoOrdenamiento orden). Es el constructor del arreglo con orden. Este arreglo, además de indicar el tamaño, debe crear un enumerado en donde se indique el orden/acomodo de las inserciones, ya sea INC (1) o DEC (2) [incremental o decremental]. Ese orden se respetará en todo el conjunto de métodos.

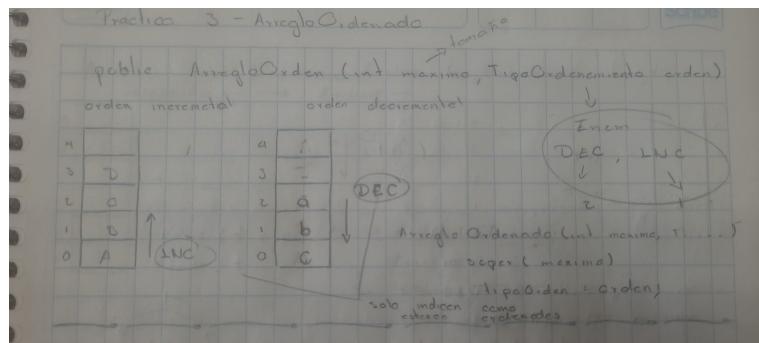


Figure 1: Análisis: mantener el valor de orden

Explicación

para poder mantener el valor de si es incremental o decremental a la clase le mandamos un valor de una enumeración que tiene dos valores, INC o DEC en el constructor, que indicaran el tipo de orden que debe tener

2.2 Análisis 2

public Integer poner(Object valor). Insertar elementos mediante mecanismos ordenados en un orden definido en el constructor.

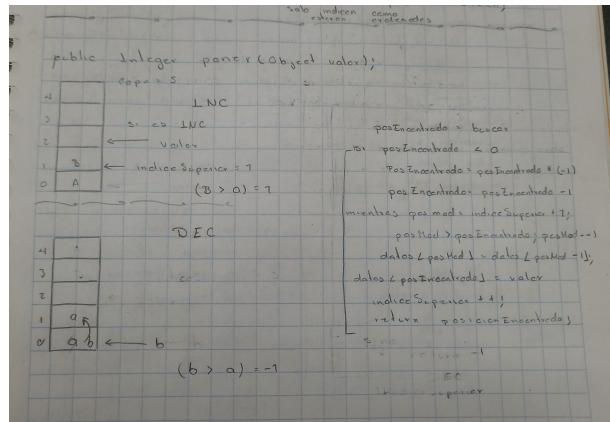


Figure 2: Análisis: agregar un objeto al arreglo

Explicación

Este fue analizado en clase, en la cual llegamos a la solución de usar un método extra de buscar, el cual nos dice si esta o no, donde si no esta nos regresa el valor donde estaría, pero si ya esta, el método no haría nada, lo que haría internamente este método es ir iterando si el objeto es mayor que el que ya esta en el arreglo, si es mayor entonces sigue iterando, pero si ya no es mayor y es igual al valor, entonces se sale del ciclo y cambia los objetos de posición, el valor de la posición lo manda un espacio arriba y en la posición agregar el objeto

2.3 Análisis 3

public Object buscar(Object valor). Buscar elementos mediante mecanismos ordenados

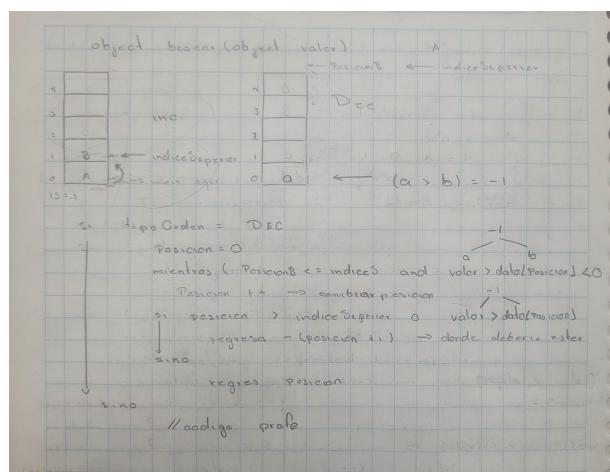
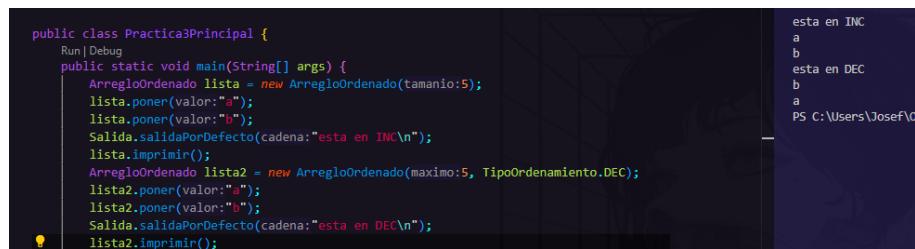


Figure 3: Análisis: buscar un objeto en un arreglo

Explicación

Este fue analizado en clase, en la cual llegamos a la solución de usar un método extra de buscar, el cual nos dice si esta o no, donde si no esta nos regresa el valor donde estaría, pero si ya esta, el método no haría nada, lo que haría internamente este método es ir iterando si el objeto es mayor que el que ya esta en el arreglo, si es mayor entonces sigue iterando, pero si ya no es mayor y es igual al valor, entonces se sale del ciclo y cambia los objetos de posición, el valor de la posición lo manda un espacio arriba y en la posición agregar el objeto



```

public class Practica3Principal {
    Run | Debug
    public static void main(String[] args) {
        ArregloOrdenado lista = new ArregloOrdenado(tamano:5);
        lista.poner(valor:"a");
        lista.poner(valor:"b");
        Salida.salidaPorDefecto(cadena:"esta en INC\n");
        lista.imprimir();
        ArregloOrdenado lista2 = new ArregloOrdenado(maximo:5, TipoOrdenamiento.DEC);
        lista2.poner(valor:"a");
        lista2.poner(valor:"b");
        Salida.salidaPorDefecto(cadena:"esta en DEC\n");
        lista2.imprimir();
    }
}
    
```

Figure 4: Análisis: funcionamiento del método poner, buscar y constructores

2.4 Análisis 4

public boolean modificar(int indice, Object valor). Este método deberá sobreescibirse para que el contenido del arreglo se reorganice

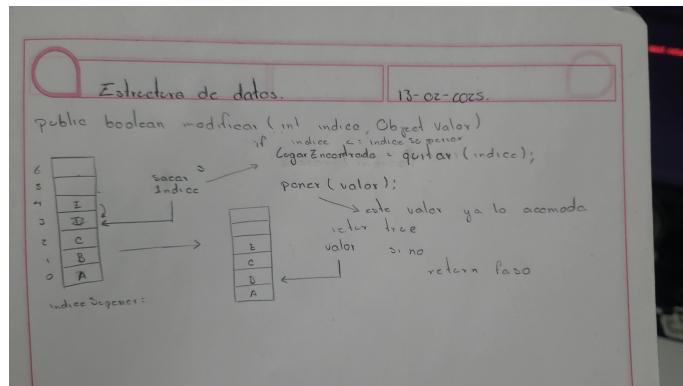


Figure 5: Análisis: modificar un dato en un arreglo

Explicación para este metodo lo que se hizo fue convertir el tipo de dato ListaDatos a ArregloOrdenado, en el cual lo cual tuvimos que hacer lo siguiente con una ayuda de una función auxiliar, si saberCantidadVacios() es mayor lista cantidad de valores contenidos, entonces iteramos dentro de lista2 sacando los valores contenido y los vamos agregando a la lista actual y regresamos un true, pero si la lista actual tiene menos espacios vacíos que la cantidad de elementos que contiene este arreglo entonces regresara un falso

```
ArregloOrdenado lista2 = new ArregloOrdenado(maximo:5, TipoOrdenamiento:  
lista2.poner(valor:"a");  
lista2.poner(valor:"b");  
lista2.poner(valor:"c");  
  
Salida.salidaPorDefecto(cadena:"esta en DEC\n");  
lista2.imprimir();  
Salida.salidaPorDefecto(cadena:"modificando b\n");  
lista2.modificar(indice:1, valor:"d");  
lista2.imprimir();
```

Figure 6: Análisis: funcionamiento del método modificar

2.5 Análisis 5

public boolean agregarLista(ListaDatos lista2). Debe permitir agregar los elementos de lista2 (que debe validar que sea un arreglo [ordenado o desordenado] en este caso) en el arreglo actual. Debe reorganizar todos los elementos insertados, de tal manera que el arreglo siga ordenado. Recuerde que el arreglo ordenado no permite valores duplicados. lista2 debe ser de tipo ArregloOrdenado

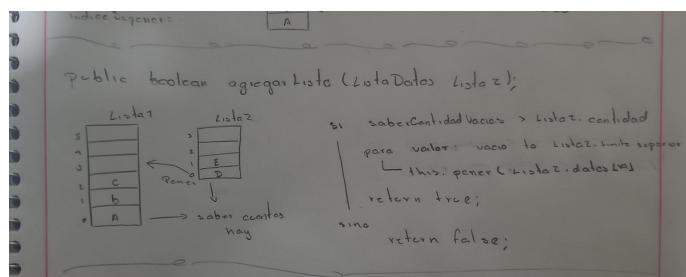


Figure 7: Análisis: agregar una lista al arreglo existente

Explicación Aquí lo que se hizo fue hacer uso de la función quitar que remueve un elemento y acomoda a los otros elementos, lo mismo se hizo con la función poner ya que agrega el elemento nuevo y lo ordena.

```
public static void main(String[] args) {
    ArregloOrdenado lista = new ArregloOrdenado(tamano:5);
    Salida.salidaPorDefecto(cadena:"lista 1");
    lista.poner(valor:"d");
    lista.poner(valor:"e");
    //Salida.salidaPorDefecto("esta en INC\n");
    //lista.imprimir();

    ArregloOrdenado lista2 = new ArregloOrdenado(maximo:5, TipoOrdenamiento.DEC);
    Salida.salidaPorDefecto(cadena:"lista 2");

    lista2.poner(valor:"a");
    lista2.poner(valor:"b");
    lista2.poner(valor:"c");

    Salida.salidaPorDefecto(cadena:"esta en DEC\n");
    lista2.imprimir();
    /*
    Salida.salidaPorDefecto("modificando b\n");
    lista2.modificar(1, "d");
    lista2.imprimir();
    */
    Salida.salidaPorDefecto(cadena:"uniendo dos listas\n");
    lista2.agregarLista(lista);
    lista2.imprimir();
}
```

Figure 8: Análisis: funcionamiento del método agregar lista

2.6 Análisis 6

[declarado en interface ListaDatos] **public void invertir()**. Debe invertir el orden de los elementos del arreglo. Al mismo tiempo que se cambia el orden, debe cambiarse el valor de la variable TipoOrdenamiento, sino mostrará inconsistencias.

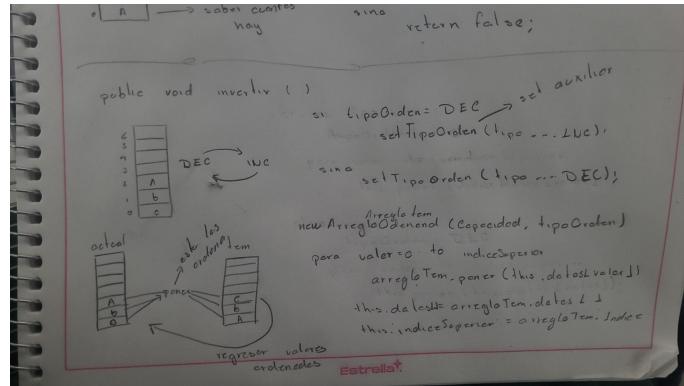


Figure 9: Análisis: invierte el orden de la lista y se re acomodo de datos

Explicación Lo que se hizo era identificar que tipo de orden era para poder cambiarlo por su inverso, para después crear un arreglo auxiliar que agregara todos los valores del arreglo actual de manera ordenada según el cambio realizado, para después pasar los valores al arreglo actual

```

@Override //se invierte el tipo de orden que tiene el arreglo
public void invertir(){
    if (tipoOrden == TipoOrdenamiento.DEC){
        setTipoOrden(TipoOrdenamiento.INC);
    } else {
        setTipoOrden(TipoOrdenamiento.DEC);
    }

    ArregloOrdenado arregloTemp = new ArregloOrdenado(capacidad, tipoOrden); //arreglo auxiliar
    for (int valor = 0; valor < indiceSuperior; valor++){
        arregloTemp.poner(this.datos[valor]);
    }
    //igualar los valores de arreglo aux con los de la lista actual
    this.datos = arregloTemp.datos;
    this.indiceSuperior = arregloTemp.indiceSuperior;
}

public void setTipoOrden(TipoOrdenamiento tipoOrden) { //get auxiliar
    this.tipoOrden = tipoOrden;
}

```

Figure 10: Análisis: funcionamiento del método invertir

2.7 Análisis 7

[declarado en interface ListaDatos] **public void invertir()**. Debe invertir el orden de los elementos del arreglo. Al mismo tiempo que se cambia el orden, debe cambiarse el valor de la variable TipoOrdenamiento, sino mostrará inconsistencias.

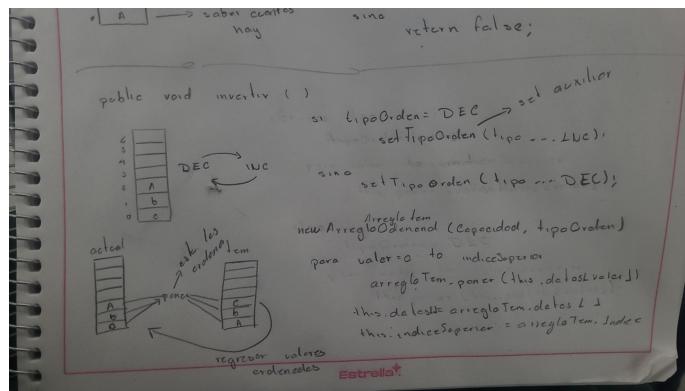


Figure 11: Análisis: invierte el orden de la lista y se re acomodo de datos

Explicación Lo que se hizo era identificar que tipo de orden era para poder cambiarlo por su inverso, para después crear un arreglo auxiliar que agregara todos los valores del arreglo actual de manera ordenada según el cambio realizado, para después pasar los valores al arreglo actual

```

@Override //se invierte el tipo de orden que tiene el arreglo
public void invertir(){
    if (tipoOrden == TipoOrdenamiento.DEC){
        setTipoOrden(TipoOrdenamiento.INC);
    } else {
        setTipoOrden(TipoOrdenamiento.DEC);
    }

    ArregloOrdenado arregloTemp = new ArregloOrdenado(capacidad, tipoOrden); //arrglo auxiliar
    for (int valor = 0; valor < indiceSuperior; valor++) {
        arregloTemp.poner(this.datos[valor]);
    }
    //igualar los valores de arreglo aux con los de la lista actual
    this.datos = arregloTemp.datos;
    this.indiceSuperior = arregloTemp.indiceSuperior;
}

public void setTipoOrden(TipoOrdenamiento tipoOrden) { //get auxiliar
    this.tipoOrden = tipoOrden;
}

```

Figure 12: Análisis: funcionamiento del método invertir

2.8 Análisis 8

[declarado en interface ListaDatos] public ListaDatos arregloDesordenado(). Debe regresar un arreglo desordenado, de tal manera que los elementos almacenados deben reburujarse a tal grado que ya no estén ordenados, no solo regresar un objeto tipo Arreglo. de los elementos del arreglo. Al mismo tiempo que se cambia el orden, debe cambiarse el valor de la variable TipoOrdenamiento, sino mostrará inconsistencias.

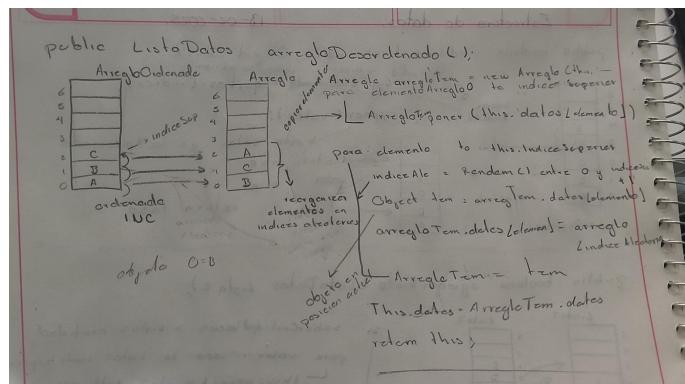


Figure 13: Análisis: de como desordenar un arreglo y pasarlo a Arreglo

Explicación Lo que se hizo fue hacer una copia del arregloOrdenado en un Arreglo normal, sobre ese mismo íbamos sacando indices aleatorios, donde primero guardamos el valor el la primera posición del arreglo, luego en esa posición guardemos el nuevo valor del mismo arreglo pero lo ponemos en el indice de donde sacamos el primer valor y para finalizar el valor que guardamos lo ponemos en el indice Aleatorio y regresamos el Arreglo normal

```

list2.imprimir();
/*
Salida.salidaPorDefecto("modificando b\n");
list2.modificar(i, "d");
list2.imprimir();
*/
Salida.salidaPorDefecto(cadena:"uniendo dos listas\n");
list2.agregarLista(lista);
list2.imprimir();

Salida.salidaPorDefecto(cadena:"cambiando orden de INC a DEC\n");
list2.invertir();
list2.imprimir();

Salida.salidaPorDefecto(cadena:"cambiando a arreglo des ordenado\n");
Arreglo arr = new Arreglo(tamano:10);
arr = (Arreglo)list2.arregloDesordenado();
arr.imprimir();

```

```

AñoSemestre\estructuraDatos\ID_46
lista lista2
b
c
uniendo dos listas
a
b
c
d
e
cambiando orden de INC a DEC
e
d
c
b
a
cambiando a arreglo des ordenado
a
b
c
d

```

Figure 14: Análisis: funcionamiento del método arregloDesordenado

2.9 Análisis 9

[declarado en interface ListaDatos] public boolean esSublista(ListaDatos lista2). Debe indicar si la lista2 (que es otro arreglo ordenado) es una sublista o subconjunto de la lista actual. Por ejemplo, la lista actual es: 1, 2, 3, 4, 5 y la lista2 es: 3, 4, 5; el valor de retorno debe ser true

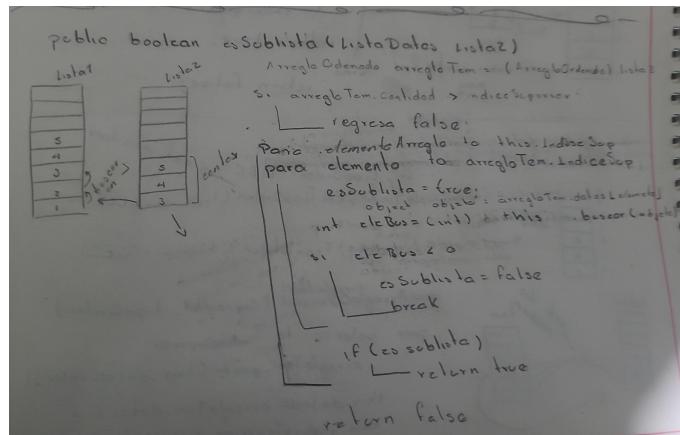


Figure 15: Análisis: de como saber si es una sublista

Explicación bueno en este caso lo que se hizo fue hacer un casting a la lista2, la cual paso a ser ArregloOrdenado, se comprobo que la cantidad no fuera igual al total de elementos en la lista1 por que eso no seria una sublista, pa siguinete entrar en una iteracion es la lista1, pero dentro de esa sentencia repetitiva lo que se hizo fu iterar dentro del rango de la lista dos para ir sacando los datos de ambas lista, las cuales se comprobaba que estuviera en la lista1 con la función búsqueda donde si nos regresa un valor menor que 0 entonces este no estaba en indicando que no es una sublista pero en caso d que si estuviera retornaba un verdadero

```
ArregloOrdenado lista = new ArregloOrdenado(tamano:5);
Salida.salidaPorDefecto(cadena:"lista 1");
lista.poner(valor:"d");
lista.poner(valor:"x");
//Salida.salidaPorDefecto("esta en INC\n");
lista.Imprimir();

ArregloOrdenado lista2 = new ArregloOrdenado(tamano:5);
Salida.salidaPorDefecto(cadena:"lista 2");

lista2.poner(valor:"a");
lista2.poner(valor:"b");
lista2.poner(valor:"c");
lista2.poner(valor:"d");
lista2.poner(valor:"e");
lista2.Imprimir();
Salida.salidaPorDefecto(cadena:"es lista 1 sub lista de lista 2 \n");

if (lista2.esSublista(lista)){
    Salida.salidaPorDefecto(cadena:"si es sublistा");
} else{
    Salida.salidaPorDefecto(cadena:"no es ");
}

PS C:\Users\Josef\OneDrive\Documentos\4tros
PS C:\Users\Josef\OneDrive\Documentos\4tros
PS C:\Users\Josef\OneDrive\Documentos\4tros
structurasBolsas\ED_40_2025 & 'C:\Program F
iles\WindowsPowerShell\Modules\Read-Host\Read-H
ost.ps1' -cp 'C:\Users\Josef\OneDrive\Docu
ments\4tros
lista 1
e
lista 2a
b
c
d
e
es lista 1 sub lista de lista 2
si es sublistा
PS C:\Users\Josef\OneDrive\Documentos\4tros
PS C:\Users\Josef\OneDrive\Documentos\4tros
PS C:\Users\Josef\OneDrive\Documentos\4tros
structurasBolsas\ED_40_2025 & 'C:\Program F
iles\WindowsPowerShell\Modules\Read-Host\Read-H
ost.ps1' -cp 'C:\Users\Josef\OneDrive\Docu
ments\4tros
lista 1
x
lista 2a
b
c
d
e
es lista 1 sub lista de lista 2
no es
PS C:\Users\Josef\OneDrive\Documentos\4tros
```

Figure 16: Análisis: funcionamiento del método esSublista

2.10 Análisis 10

[declarado en interface ListaDatos] public boolean modificarLista(ListaDatos lista2, ListaDatos lista2Nuevos). Debe cambiar los elementos de lista2 que se encuentren en la lista actual con los elementos de la lista2Nuevos. Cada elemento de lista2 coincide en posición con su nuevo valor a cambiar en lista2Nuevos. Ejemplo, lista2= 2, 3, 4, lista2Nuevos=50, 40, 80. Quiere decir que si encuentra un 2 en lista actual debe substituirlo por un 50, si encuentra un 3 en lista actual debe substituirlo por 40, etc.

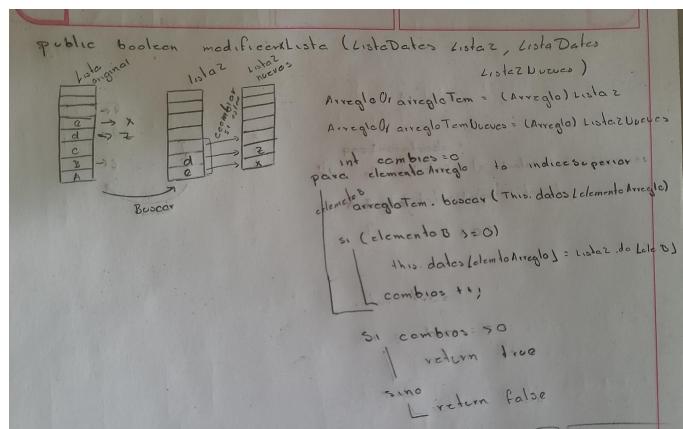


Figure 17: Análisis: de como cambiar valores de una lista por los de otra, si están en una lista secundaria

Explicación En este caso lo que se hizo fue hacer un castin para las dos listas que recibimos, las cuales las pasamos a tipo Arreglo, iteramos en sobre la lista actual para ir sacando los datos y ir buscándolos en la lista2 y en caso de si estar lo que pasa es que cambiamos el dato en la posición del dato de la lista actual, por el dato de la lista2Nuevos, en la posición del elemento en la lista2 y a un contador le sumamos 1, para después de hacer esto con todo los datos preguntarle al contador si hubo cambios y si si hubo regresa true y si no false

```

ListaDatos lista = new ArregloOrdenado(tamano:5);
Salida.salidaPorDefecto(cadena:"lista 1\n");
lista.poner(valor:"d");
lista.poner(valor:"e");
//Salida.salidaPorDefecto("esta en INC\n");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"lista nuevos objetos\n");
ListaDatos lista3 = new ArregloOrdenado(tamano:5);
lista3.poner(valor:"a");
lista3.poner(valor:"b");
lista3.poner(valor:"c");
lista3.poner(valor:"d");
lista3.poner(valor:"e");
lista3.imprimir();

ArregloOrdenado lista2 = new ArregloOrdenado(tamano:5);
Salida.salidaPorDefecto(cadena:"lista 2 donde se cambia los valores \n");

lista2.poner(valor:"a");
lista2.poner(valor:"b");
lista2.poner(valor:"c");
lista2.poner(valor:"d");
lista2.poner(valor:"e");
lista2.imprimir();
Salida.salidaPorDefecto(cadena:"cambiando valores \n");

if [lista2.modificarLista(lista, lista3)][]
    Salida.salidaPorDefecto(cadena:"si se cambio\n");
else
    Salida.salidaPorDefecto(cadena:"no se cambio\n");
}
lista2.imprimir();

```

PRINCIPAL
 lista 1
 d
 e
 lista nuevos objetos
 a
 b
 c
 d
 e
 cambiando valores
 si se cambio
 a
 b
 c
 x
 z
 PS C:\Users\Josef\OneDrive\Documentos\V4

Figure 18: Análisis: funcionamiento del método modificar lista con parámetros tipo ListaDatos

2.11 Análisis 11

public boolean retenerLista(ListaDatos lista2). Debe dejar en la lista actual solo los elementos que se encuentran en lista2.

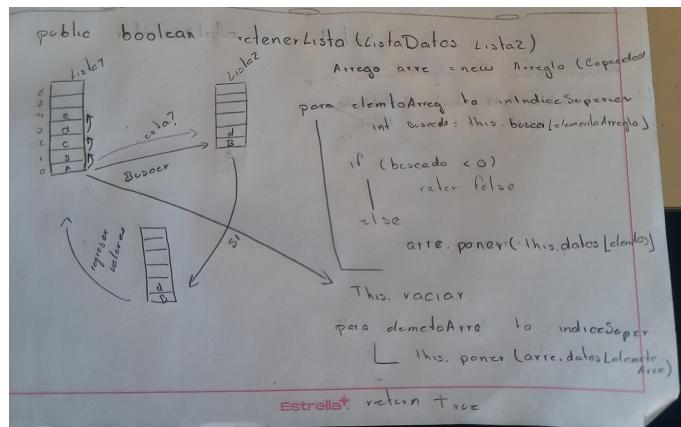
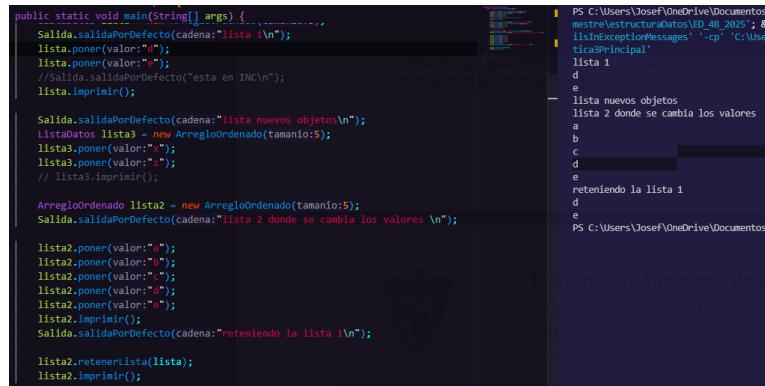


Figure 19: Análisis: de método que deja los elementos que pertenezcan a un arreglo secundario

Explicación Aquí lo que se hizo fue un casting para el parámetro, además creamos un arreglo auxiliar, además de eso se itera sacando los datos del arreglo principal y buscándolos en el arreglo secundario, donde si el valor es mayor o igual a cero, este dato se va a agregar en el arreglo secundario y así hasta acabar con la lista original, una vez con los datos de la lista secundaria aparte, se vacía la lista original, para después volver a insertarle únicamente los datos de la lista secundaria



```

public static void main(String[] args) {
    Salida.salidaPorDefecto(cadena:"lista 1\n");
    lista1.poner(valor:"a");
    lista1.poner(valor:"b");
    lista1.poner(valor:"c");
    lista1.poner(valor:"d");
    lista1.poner(valor:"e");
    lista1.imprimir();

    Salida.salidaPorDefecto(cadena:"lista nuevos objetos\n");
    ListaDatos lista3 = new ArregloOrdenado(tamano:5);
    lista3.poner(valor:"x");
    lista3.poner(valor:"y");
    lista3.poner(valor:"z");
    // lista3.imprimir();

    ArregloOrdenado lista2 = new ArregloOrdenado(tamano:5);
    Salida.salidaPorDefecto(cadena:"lista 2 donde se cambia los valores \n");

    lista2.poner(valor:"a");
    lista2.poner(valor:"b");
    lista2.poner(valor:"c");
    lista2.poner(valor:"d");
    lista2.poner(valor:"e");
    lista2.imprimir();
    Salida.salidaPorDefecto(cadena:"reteniendo la lista 1\n");

    lista2.retenereLista(lista1);
    lista2.imprimir();
}

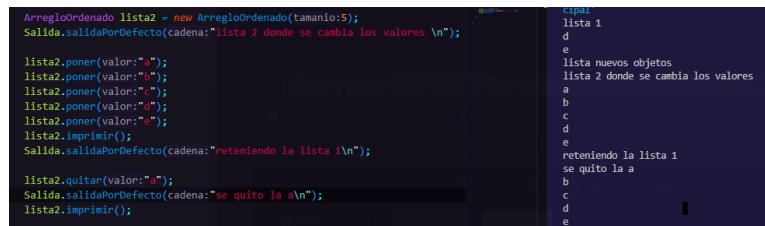
```

Figure 20: Análisis: funcionamiento del método retener datos

2.12 Análisis 12

public Object quitar(Object valor). Eliminar un elemento del arreglo

Explicación En este método podemos ver como se remueve elementos del Arreglo-Ordenado por medio de un parámetro, el cual es el elemento a eliminar, que se usa con la búsqueda para saber su posición y eliminarlo, desplazando a todos los elementos por arriba de ella a un espacio abajo para tener un arreglo sin huecos o vacíos



```

ArregloOrdenado lista2 = new ArregloOrdenado(tamano:5);
Salida.salidaPorDefecto(cadena:"lista 2 donde se cambia los valores \n");

lista2.poner(valor:"a");
lista2.poner(valor:"b");
lista2.poner(valor:"c");
lista2.poner(valor:"d");
lista2.poner(valor:"e");
lista2.imprimir();
Salida.salidaPorDefecto(cadena:"reteniendo la lista 1\n");

lista2.quitar(valor:"a");
Salida.salidaPorDefecto(cadena:"se quito la a\n");
lista2.imprimir();

```

Figure 21: Análisis: funcionamiento de eliminar objeto

Actividad 2

Complete el TDA llamado “Arreglo”. Para esto deberá codificar el método:

2.13 Análisis 13

[declarado en interface ListaDatos] public boolean poner(int indice, Object info).

Este método insertará en la posición “indice” el contenido de valor. También modifique el método para sea sobre-escrito en la clase ArregloOrdenado de tal manera que se valide que siga ordenado.

- También modifique el método para sea sobre-escrito en la clase ArregloOrdenado de tal manera que se valide que siga ordenado.

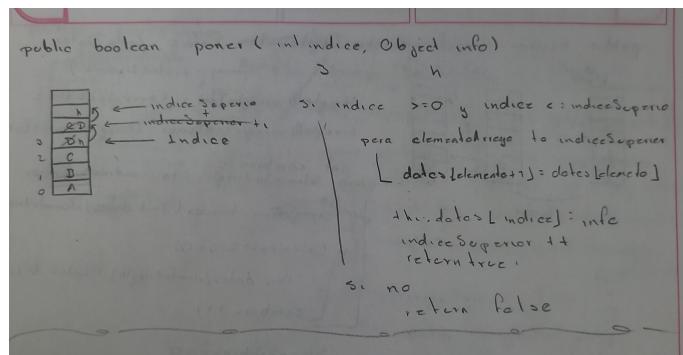


Figure 22: Análisis: método de poner en un índice específico

Explicación En este método lo que se hizo fue revisar que el índice dado estuviera dentro del rango, ya que no queremos espacios vacíos, si esto no pasa, entonces el método retornar false, lo siguiente por si si cumple la condición es iterar en la lista de manera decremental para poder ir moviendo lo objeto una posición arriba para liberar el espacio del índice y poder insertar el elemento, para después sumarle unos al índice y retornar true

```

public boolean poner ( int indice, Object info )
{
    if ( indice >= 0 && indice < indiceSuperior )
        for ( elemento = indice + 1 ; elemento < indiceSuperior ;
              elemento++ )
            datos [ elemento + 1 ] = datos [ elemento ];
    datos [ indice ] = info;
    indiceSuperior++;
    return true;
}

ArregloOrdenado lista1 = new ArregloOrdenado ( tamanio:5 );
Salida.salidaPorDefecto ( cadena:"lista 1\n" );
lista1.poner ( valor:"a" );
lista1.poner ( valor:"b" );
// Salida.salidaPorDefecto ("esta en INC\n");
lista1.imprimir();

Salida.salidaPorDefecto ( cadena:"lista nuevos objetos\n" );
ListaDatos lista2 = new ArregloOrdenado ( tamanio:5 );
Salida.salidaPorDefecto ( cadena:"lista 2 donde se cambia los valores \n" );

lista2.poner ( valor:"a" );
lista2.poner ( valor:"b" );
lista2.poner ( valor:"c" );
lista2.poner ( valor:"d" );
// lista2.poner ("e");
lista2.imprimir();
Salida.salidaPorDefecto ( cadena:"reteniendo la lista 1\n" );

// lista2.quitar ("a");
lista2.poner ( indice:2, info:"h" );
Salida.salidaPorDefecto ( cadena:"agregando h\n" );
lista2.imprimir();
    
```

Figure 23: Análisis: funcionamiento del método poner con índice específico

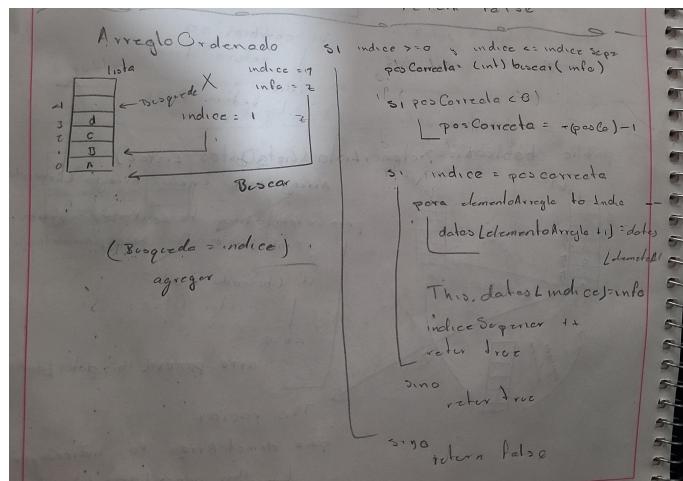


Figure 24: Análisis: método de poner en un índice específico en Arreglo ordenado

Explicación Aquí lo que se hizo fue obtener la posición de donde debería de estar, y comparar con el índice y si estas dos son iguales procedemos a recorrer los elementos, y después podremos meter el elemento deseado



```

public static void main(String[] args) {
    lista1.poner(valor:"a");
    lista1.poner(valor:"b");
    lista1.poner(valor:"c");
    lista1.poner(valor:"d");
    lista1.poner(valor:"e");
    lista2.poner(valor:"a");
    lista2.poner(valor:"c");
    lista2.poner(valor:"d");
    lista2.poner(valor:"e");
    lista2.poner(valor:"b");
    lista2.imprimir();
    Salida.salidaPorDefecto(cadena:"lista 2 donde se cambia los valores \n");

    lista2.poner(valor:"a");
    lista2.poner(valor:"c");
    lista2.poner(valor:"d");
    lista2.poner(valor:"e");
    lista2.poner(valor:"b");
    lista2.imprimir();
    Salida.salidaPorDefecto(cadena:"reteniendo la lista 1\n");

    //lista2.quitar("");
    lista2.poner(indice:1, info:"b");
    Salida.salidaPorDefecto(cadena:"agregando b\n");
    lista2.imprimir();
}
    
```

Figure 25: Análisis: funcionamiento del método poner con índice específico

2.14 Análisis 14

[declarado en interface ListaDatos] **public boolean substituir(ListaDatos lista2)**. Copiar el contenido de lista2 a la lista actual. El contenido de la lista actual se perderá al ser substituido por la lista2. Este método ya lo debe tener de la práctica

- Modifique el método para sea sobre-escrito en la clase ArregloOrdenado tal manera que se valide que siga ordenado. Es decir, si tiene un orden definido en lista2, se debe poder copiar si se respeta el orden INC o DEC, en caso contrario no debe poderse hacer.

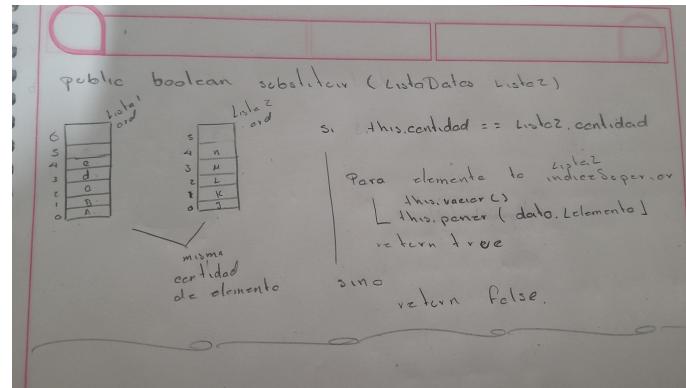
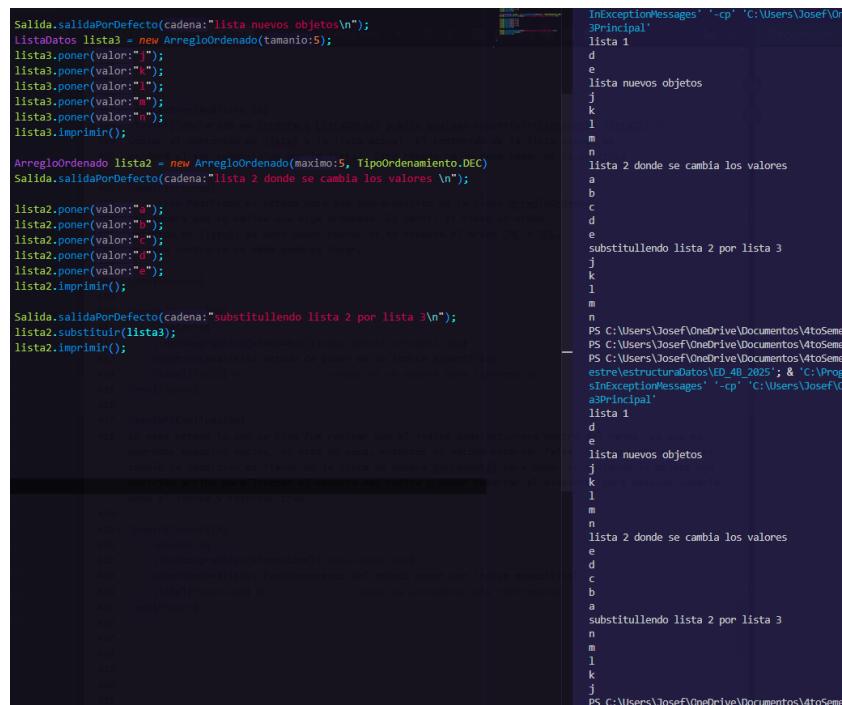


Figure 26: Análisis: método de substituir pero según un orden específico

Explicación Aquí lo que se hizo fue tomar las bases del método en la clase Arreglo que para poder sustituir una lista con otra, hay que tener en ambas listas la misma cantidad de elementos para poder remplazar la totalidad de los elementos



```

Salida.salidaPorDefecto(cadena:"lista nuevos objetos\n");
ListaDatos lista3 = new ArregloOrdenado(tamano:5);
lista3.poner(valor:"j");
lista3.poner(valor:"k");
lista3.poner(valor:"l");
lista3.poner(valor:"m");
lista3.poner(valor:"n");
lista3.imprimir();

ArregloOrdenado lista2 = new ArregloOrdenado(maximo:5, TipoOrdenamiento.DEC)
Salida.salidaPorDefecto(cadena:"lista 2 donde se cambia los valores \n");

lista2.poner(valor:"a");
lista2.poner(valor:"b");
lista2.poner(valor:"c");
lista2.poner(valor:"d");
lista2.poner(valor:"e");
lista2.imprimir();

Salida.salidaPorDefecto(cadena:"substituyendo lista 2 por lista 3\n");
lista2.substituir(lista3);
lista2.imprimir();
  
```

Terminal output:

```

InExceptionMessages" "-cp" "C:\Users\Josef\OneDrive\Documentos\4toSemestre\Práctica 1\Java\estructuraDatos\TD_4B_2025" & C:\Program Files\Java\jdk-17.0.2\bin\java -jar a3PPrincipal.jar
lista 1
d
e
lista nuevos objetos
j
k
l
m
n
lista 2 donde se cambia los valores
a
b
c
d
e
substituyendo lista 2 por lista 3
j
k
l
m
n
PS C:\Users\Josef\OneDrive\Documentos\4toSemestre\Práctica 1\Java\estructuraDatos\TD_4B_2025> & C:\Program Files\Java\jdk-17.0.2\bin\java -jar a3PPrincipal.jar
lista 1
d
e
lista nuevos objetos
j
k
l
m
n
lista 2 donde se cambia los valores
e
d
c
b
a
substituyendo lista 2 por lista 3
n
m
l
k
j
PS C:\Users\Josef\OneDrive\Documentos\4toSemestre\Práctica 1\Java\estructuraDatos\TD_4B_2025>
  
```

Figure 27: Análisis: funcionamiento del método poner con indice específico

Actividad 4

Defina un programa en Java que permita guardar los datos de un índice de términos/subtérminos de un libro (habitualmente vienen al final de un libro, y sirve para consultar o encontrar fácilmente términos en él), así como sus páginas donde aparece (se incluye un ejemplo en el archivo anexo). Se le pide que estos datos sean agregados en orden basado en nombre.

Genere un menú de opciones con las siguientes consideraciones

- Agregar términos, subtérminos y sus páginas (éstas pueden ser individuales o rangos).
- Consultar un término con base en su descripción/nombre.
- Consultar un subtérmino con base en su descripción/nombre.
- Listar todos los términos/subtérminos y sus páginas.
- Listar solo los términos de un rango de letras iniciales, por ejemplo, solo los que inicien con B y D.

Posibles clases

- Término
- GestorTerminos
- clasePrincipal

3 Código Agregado - UML

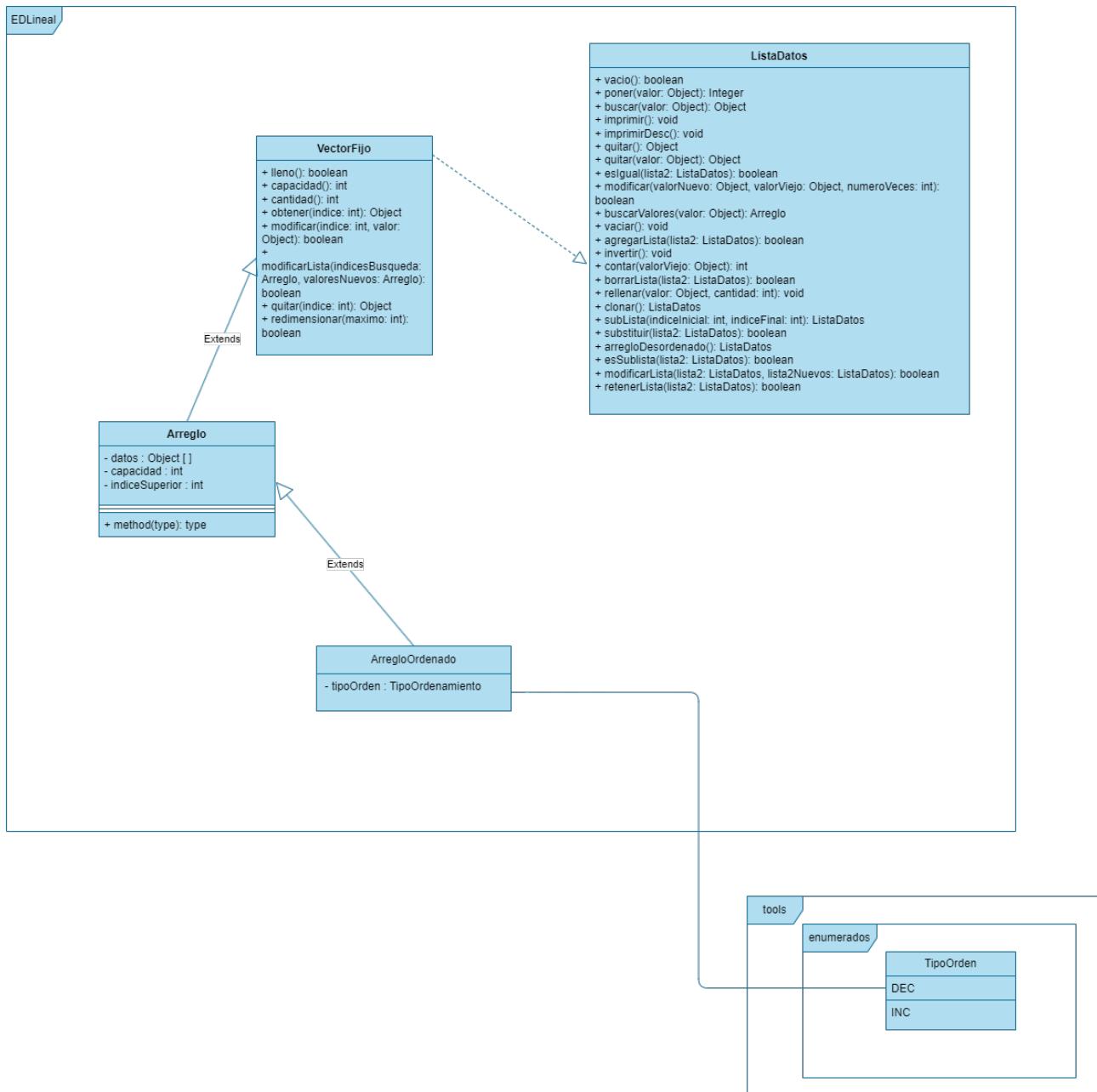


Figure 28: UML

4 Pre-evaluación del Alumno

Práctica NO: 3

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)	85%

Table 1: Evaluación de la práctica

5 Conclusión

se trabajo de manera casi intuitiva los arreglos ordenados, aun que me gustaría que fuera un poco mas claro de que es lo que se necesita cumplir con los métodos ya que en la actividad 3 me quede un poco pensante de que tenia que hacer el método 1, aun me quedan un poco de dudas en cuanto algunas cosas de herencia y implementación