Practica 3: Arreglos unidimensionales

Nombre: Jaime Gabriel Robles Félix

Grado y grupo: “4-B”

índice

1. Introducción 3
2. Desarrollo 3

2.1 Métodos y Funcionalidades 3

2.2 Constructor ArregloOrden 3

2.3 Método poner() 3

2.4 Método buscar() 6

2.5 Método modificar() 7

2.6 Método quitar() 8

2.7 Método agregarLista() 9

2.8 Método invertir() 10

2.9 Método rellenar() 11

2.10 Método arregloDesordenado() 12

2.11 Método esSublista() 13

2.12 Método modificarLista() 14

2.13 Método retenerLista() 15

Actividad 3

3.1 Método poner() 16

3.2 Método substituir() 17

Diagrama UML 18

Pre-Evaluación 19
conclusión 20

Introducción (Actividad 1)

En esta práctica aremos mucho énfasis en los arreglos unidimensionales y ordenados, ya que al usar los métodos correctos se podría sacar mucho provecho a estas, estos ayudan a varias problemáticas de forma sencilla si se plantea bien el método y la lógica.

Desarrollo (Actividad 2)

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.

   En este primer problema genere en el paquete tools, una nueva herramienta llamada TipoOrden, en esta clase de tipo enum guarda los atributos de DEC y INC, con una variable orden y un constructor el cual almacenara el atributo a la variable orden.

```
public ArregloOrdenado(int tamanio, TipoOrden orden){  no usages  new *
    super(tamanio);
}
```

Imagen 1. Este es nuestro nuevo constructor el cual recibe el TipoOrden orden

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

   Este método lo realizamos en clase.

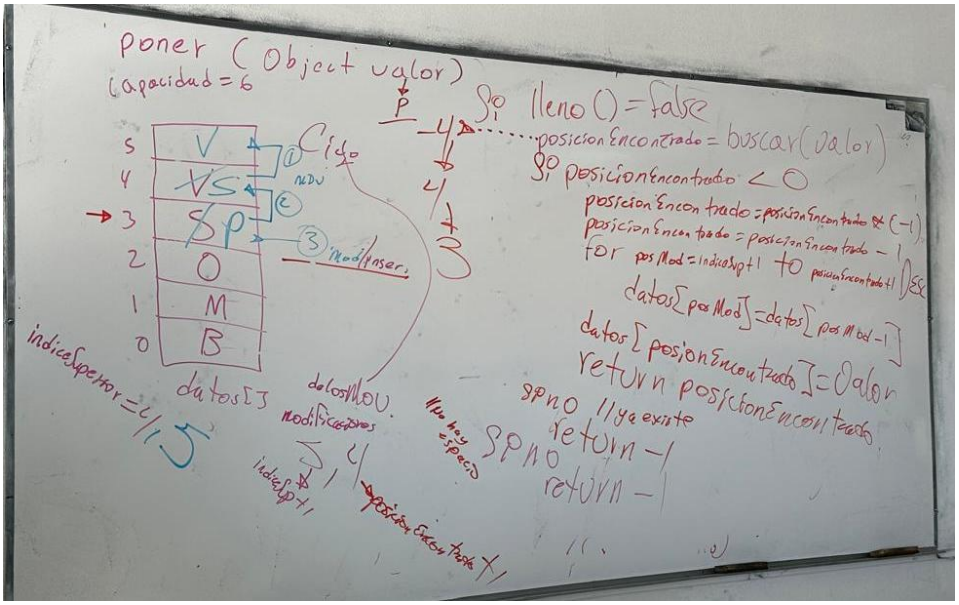


Imagen 2. Análisis y diseño en clase

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

   Este método también lo realizamos en clase presencial, de forma INCREMENTAL.

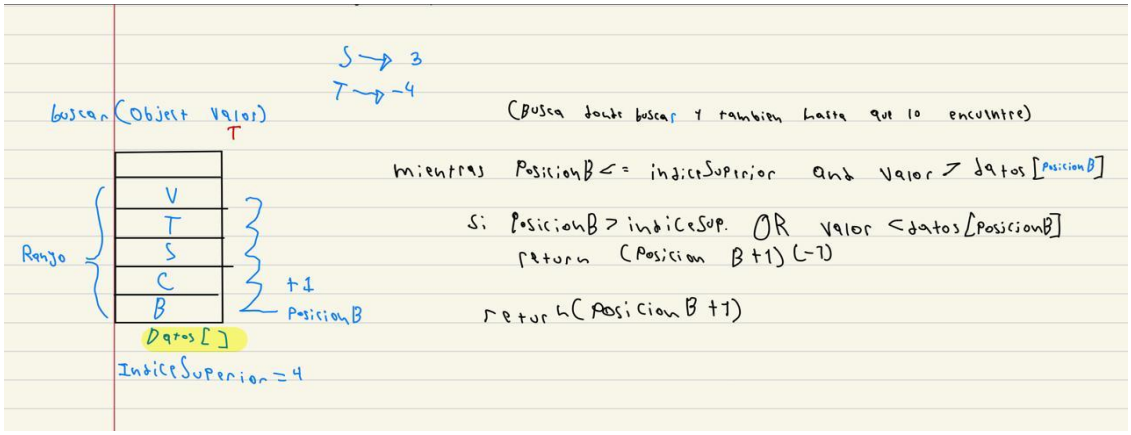


Imagen 3. Diseño de la forma incremental

- INCREMENTAL se utiliza la herramienta de comparador de objetos la cual se compara los objetos y regresa el resultado en forma de numero positivo si el objeto 1 es mayor que el objeto 2, si el objeto 1 es menor que el objeto 2 manda un negativo y si son iguales manda cero, el resultado de esta comparación determina que por cada iteración en el cual SI el objeto 1 es mayor que el objeto 2 siga con el siguiente valor, así hasta encontrar que el objeto 1 sea menor que el objeto 2, ahí ya encontró la posición de donde debe de ir el objeto a insertar.

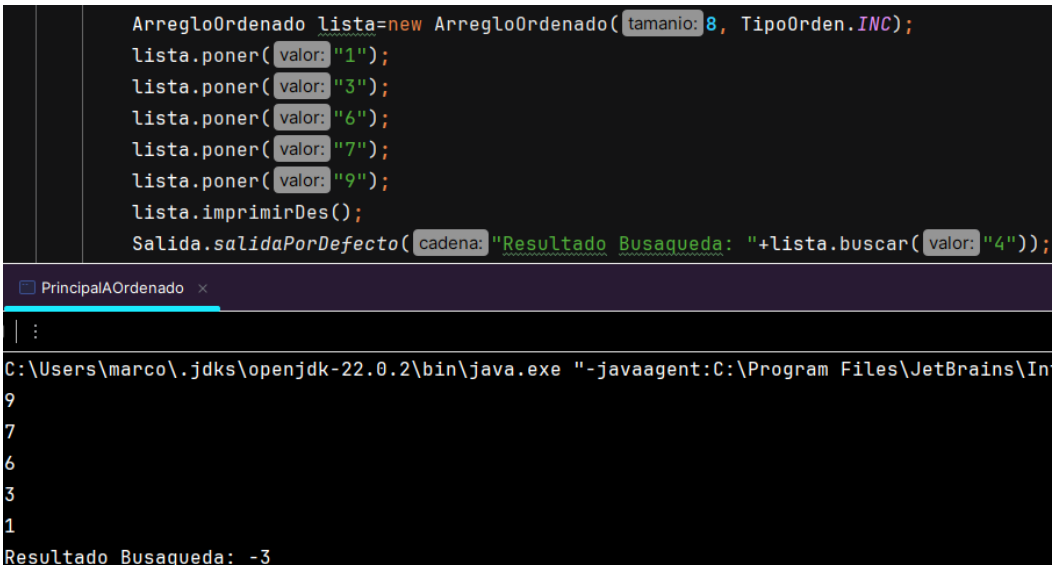


Imagen 4. Resultado de la función INC cuando buscamos "4"

- DECREMENTAL es el mismo proceso solo que el valor de las comparaciones es al revés ya que nuestra tabla es decremental, SI objeto 1 es menor a objeto 2 sigue con el siguiente objeto en la tabla hasta que encuentre que el objeto 1 es mayor al objeto 2.

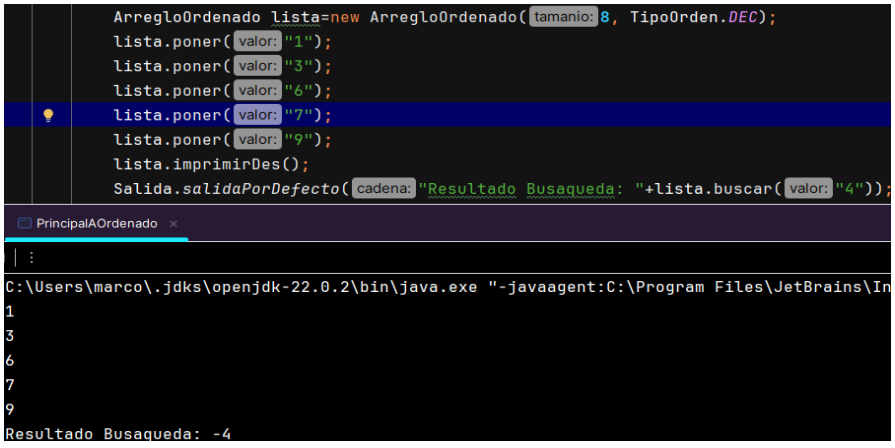


Imagen 5. Resultado de la función DEC cuando buscamos "4"

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

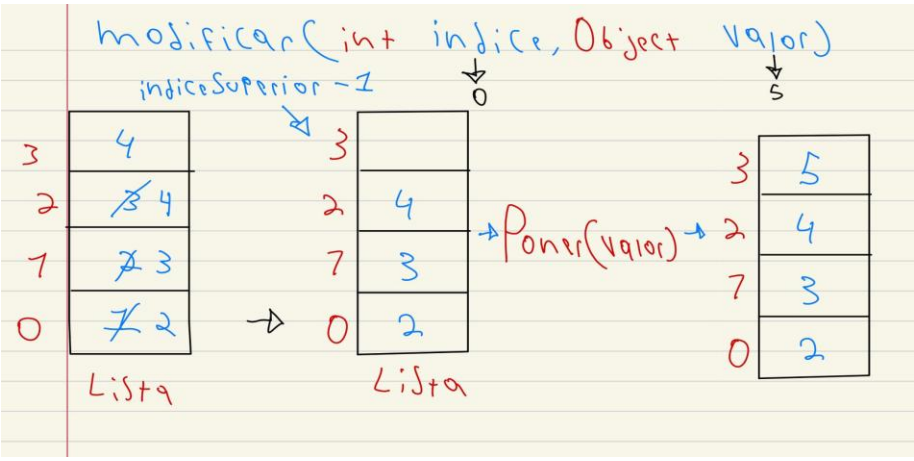


Imagen 6. Diseño de modificar

- Validamos el índice para que no tenga ningún tipo de error.
- Desde el índice recorremos el arreglo hacia abajo eliminando el valor a modificar
- Con el método poner, ingresamos el nuevo valor reorganizando la lista

```
ArregloOrdenado lista=new ArregloOrdenado( tamanio: 8, TipoOrden.INC);
lista.poner( valor: "1");
lista.poner( valor: "3");
lista.poner( valor: "6");
lista.poner( valor: "7");
lista.poner( valor: "9");
lista.modificar( indice: 2, valor: 5);
lista.imprimirDes();
```

```
PrincipalAOrdenado
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\Je
9
7
5
3
1
```

Imagen 7. Resultado del método

En el ejemplo el índice “2” es el valor “6”, pero al usar el método el valor se quita, se ingresa el nuevo valor y se reordena la lista.

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

- Primero se usa el método de **Buscar(Valor)** para que retorne la posición del objeto a quitar.
- Se **valida la posición** ya que si no encuentra el valor en la lista, retornara un índice negativo.
- Se **respalda** el objeto que se va a eliminar.
- Usando un for **recorremos** el arreglo para quitar nuestro valor.
- Le restamos uno al **índice superior** indicando que hay un espacio disponible en nuestra tabla.
- Retornamos nuestra variable de respaldo.

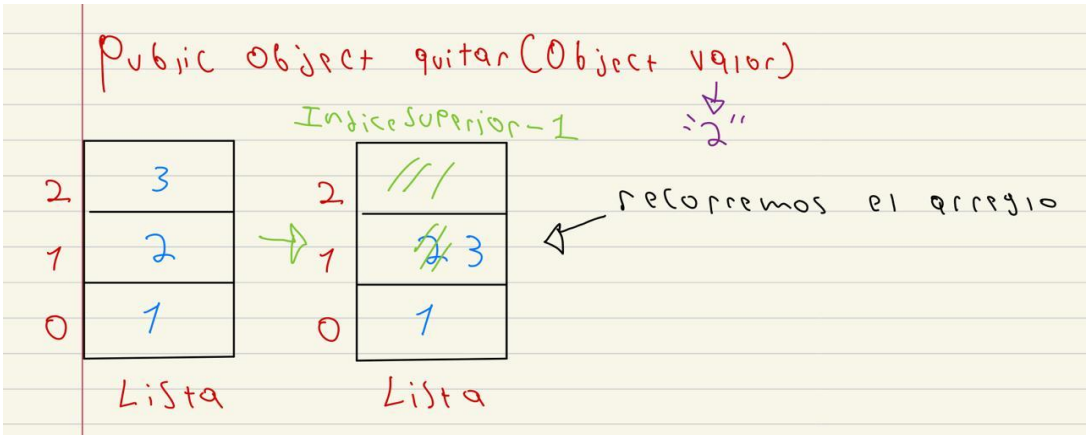


Imagen 8. Diseño del método quitar

```
ArregloOrdenado lista=new ArregloOrdenado( tamanio: 8, TipoOrden.INC);
lista.poner( valor: "1");
lista.poner( valor: "2");
lista.poner( valor: "3");
lista.poner( valor: "4");
lista.poner( valor: "5");
lista.quitar( valor: "3");
lista.imprimirDes();
```

```
PrincipalAOrdenado
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\Je
5
4
2
1
```

Imagen 9. Salida usando el método quitar

6. **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.

    - Creamos un método para convertir de **ListaDatos a ArregloOrdenado**.
    - Validamos que el arreglo convertido sea un **instance of ArregloOrdenado**.
    - Realizamos un **for** el cual recorre nuestra lista2 y con el método **poner()** inserte los datos de la tabla2 a la tabla 1 y los ordene.
    - Si todo salió bien retornamos un true y en caso de que alguna validación no se cumpla se retorna un false.

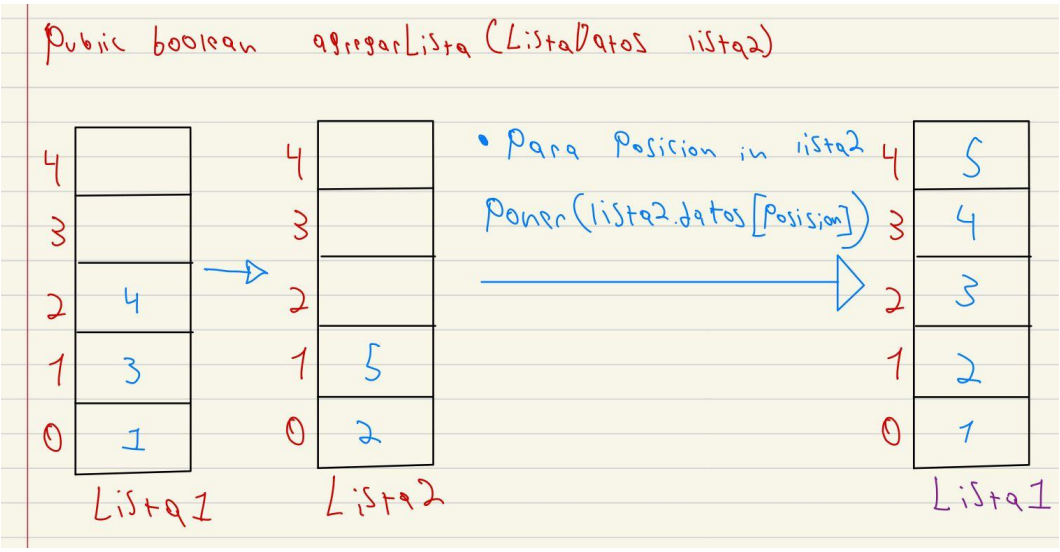


Imagen 10. Diseño de Agregar Lista

```
ArregloOrdenado lista=new ArregloOrdenado( tamanio: 8, TipoOrden.INC);
lista.poner( valor: "1");
lista.poner( valor: "6");
lista.poner( valor: "3");
ArregloOrdenado lista2=new ArregloOrdenado( tamanio: 3,TipoOrden.INC);
lista2.poner( valor: "4");
lista2.poner( valor: "7");
lista2.poner( valor: "5");
lista.agregarLista(lista2);
lista.imprimirDes();
```

```
PrincipalAOrdenado
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrai
7
6
5
4
```

Imagen 11. Salida usando el método Agregar Lista

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.

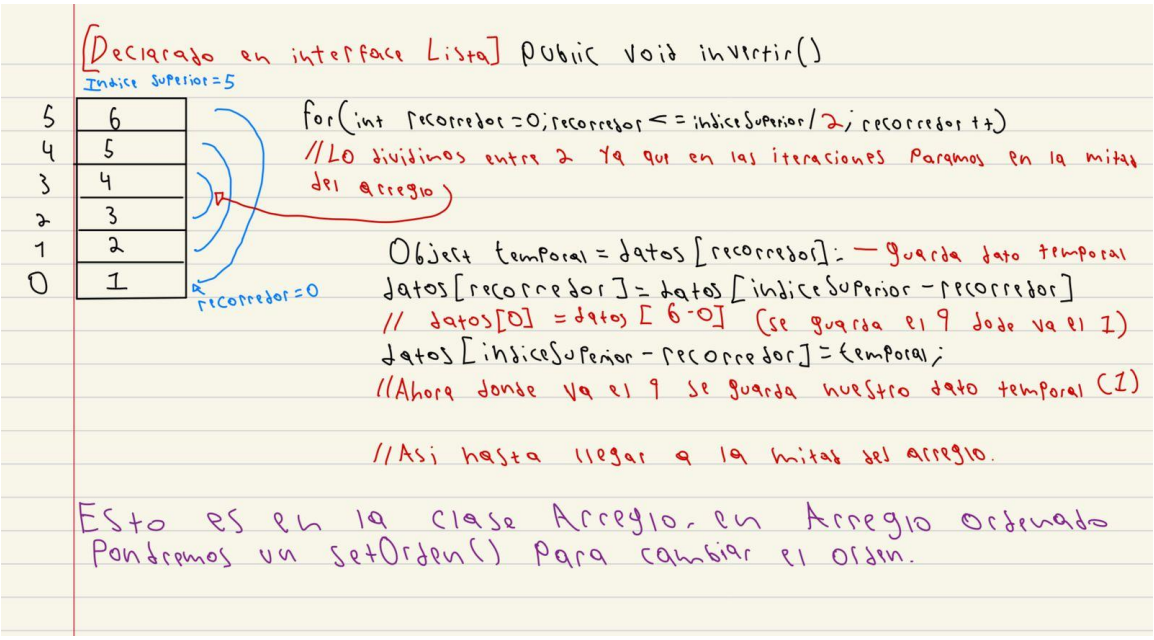


Imagen 12. Análisis y diseño del método invertir.

```
ArregloOrdenado lista=new ArregloOrdenado( tamanio: 8, TipoOrden.INC);
lista.poner( valor: "1");
lista.poner( valor: "2");
lista.poner( valor: "3");
lista.poner( valor: "4");
lista.poner( valor: "5");
lista.invertir();
lista.imprimirDes();
```

```
PrincipalAOrdenado
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetB
1
2
3
4
5
```

Imagen 13. Salida utilizando el método invertir

8. [declarado en interface ListaDatos] public void rellenar(Object valor). "valor" indica el límite superior en los valores a rellenar, siempre y cuando sea numérico. Por ejemplo, si valor es 6, se insertaron elementos del 1 al 6 (en las posiciones de 0, 1, 2, 3, 4, 5) [orden INC]. Si valor es 6 y orden es DEC, se insertaron elementos del 6 al 1 (en las posiciones de 0, 1, 2, 3, 4, 5). Si el valor es negativo, por ejemplo-6, se debe insertar elementos del-1 al-6 (en las posiciones 0, 1, 2, 3, 4, 5) [orden DEC]. Si el valor es negativo y el orden es INC, se debe insertar elementos del-6 al-1 (en las posiciones 0, 1, 2, 3, 4, 5). Si es una letra, por ejemplo H, se deberán insertar elementos como A, B, C, D, E, F, G, H (en las posiciones 0, 1, 2, 3, 4, 5….) [orden INC]. Si es una letra, por ejemplo H, se deberán insertar elementos como H, G, F, E, D, B, A (en las posiciones 0, 1, 2, 3, 4, 5….) [orden DEC]. Si es cadena, por ejemplo "hola", debe insertar solo este elemento [orden INC]. Si es otro tipo de objeto, también debe insertar solamente ese valor de manera única, ya que no se permiten valores duplicados. Recuerde que este método está en función del tamaño del arreglo, no debe sobrepasarse.

9. [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.

La visión principal es tomar un dato y con un random para el índice, intercambiar los valores de los datos.

- Primero creamos un nuevo Arreglo con el nombre "arregloDesordenado".
- Creamos el new random para apoyarnos.
- Generamos un for el cuál empiece en recorredor=0 y termine en el índice superior, dentro de él lo que hará es que toma el valor de la posición recorredor y con un random generamos un numero aleatorio de índice (Le agregamos los rangos validos) después intercambiamos los datos del índice recorredor al índice random, así por cada iteración.
- Al final retornaremos un arregloDesordenado.

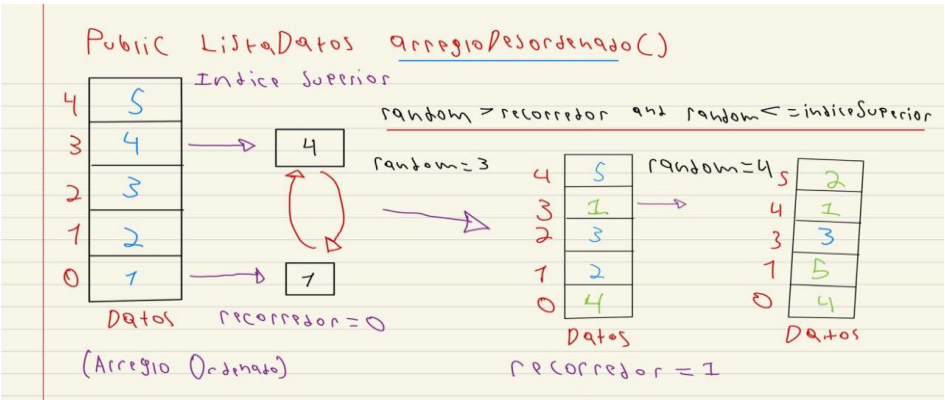


Imagen 14. Diseño método arreglo desordenado

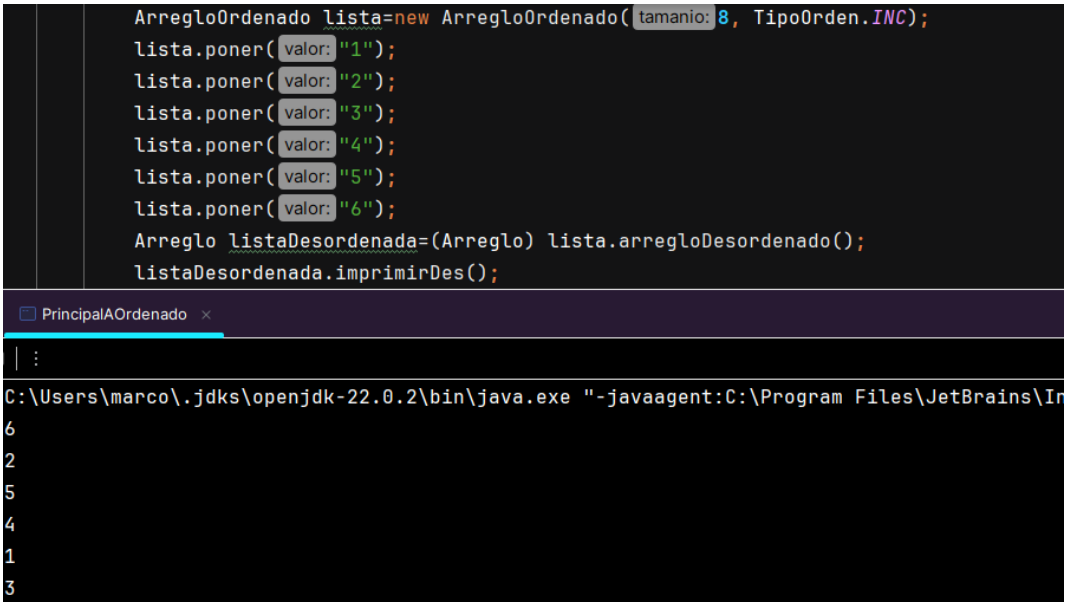


Imagen 15. Salida de lista Desordenada

10. [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.

    La visión General es que validaremos que cada dato en lista 2 sea igual a algún dato en nuestra lista actual.

    - Validamos que la lista2 sea un Arreglo Ordenado.
    - Transformamos de ListaDatos a ArregloOrdenado.
    - Usamos un for para recorrer la lista2, adentro de este for utilizaremos el metodo buscar() por cada dato de la lista, SI el resultado de la búsqueda es positivo (Significa que encontró el dato en la lista actual) sigue a la siguiente iteración, pero SI es negativo (Significa que NO encontró el dato en la lista actual) retorna un falso.

- Si recorre la lista y todos los datos fueron encontrados, retorna un **true** para validar que si es una **sublista**.

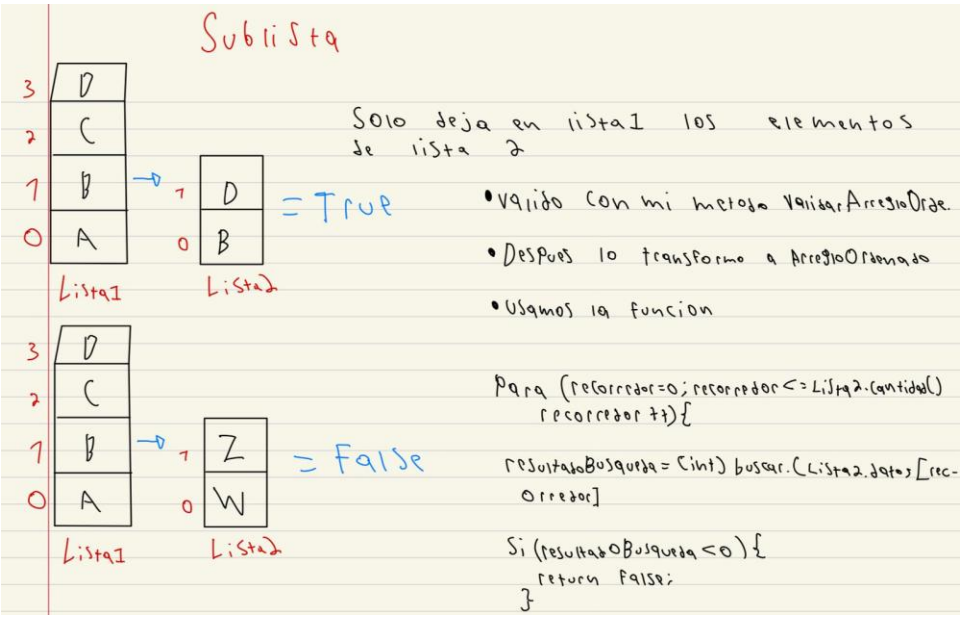


Imagen 16. Diseño del método subLista

```
ArregloOrdenado lista=new ArregloOrdenado( tamanio: 8, TipoOrden.INC);
lista.poner( valor: "1");
lista.poner( valor: "2");
lista.poner( valor: "3");
lista.poner( valor: "4");
lista.poner( valor: "5");
ArregloOrdenado lista2=new ArregloOrdenado( tamanio: 3,TipoOrden.INC);
lista2.poner( valor: "2");
lista2.poner( valor: "3");
lista2.poner( valor: "4");
Salida.salidaPorDefecto( cadena: ""+lista.esSubLista(lista2));
```

```
PrincipalAOrdenado
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrai
true
```

Imagen 17. Salida del método esSublista

11. [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.

La **visión General** es que si en la lista actual hay algún dato igual que en la lista 2, se sustituye por el valor de la lista 2 nuevos en el índice del valor que se encontró de la lista 2.

- Hacemos casting para que ListaDatos sea Arreglo
- Valida que la lista 2 y la lista 2 Nuevos tengan la misma cantidad de datos
- Hacemos un for el cual recorra la lista 2, adentro lo que va a hacer es que buscara el dato de la lista 2 en la lista 1, si lo encuentra toma el índice de la lista 2 y sustituye el valor de la lista actual por el valor del índice de la lista 2 nuevos.
- Retorna true si todo se realizó con éxito.

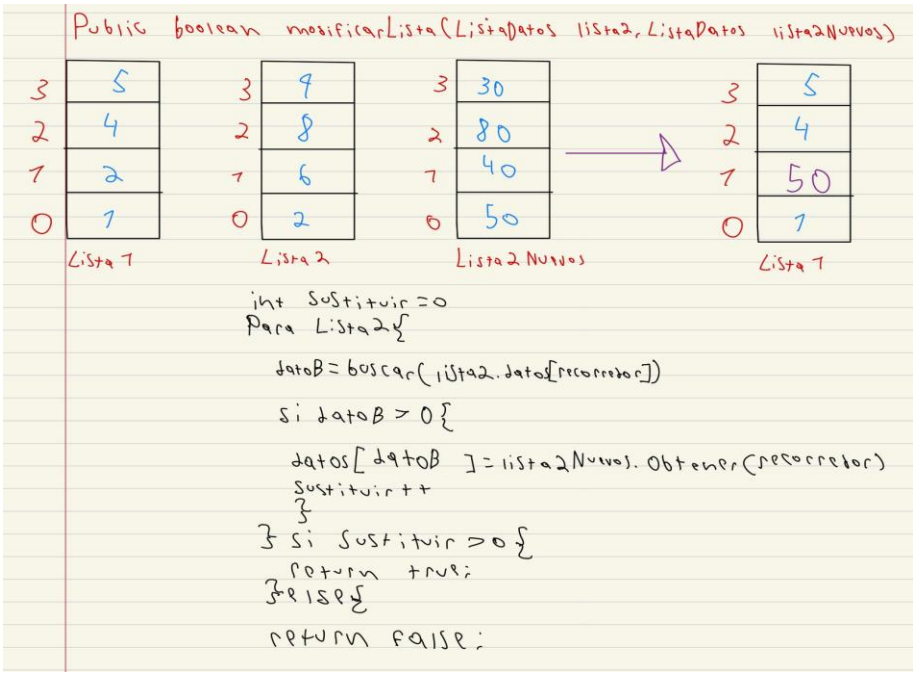


Imagen 18. Diseño del método modificar lista

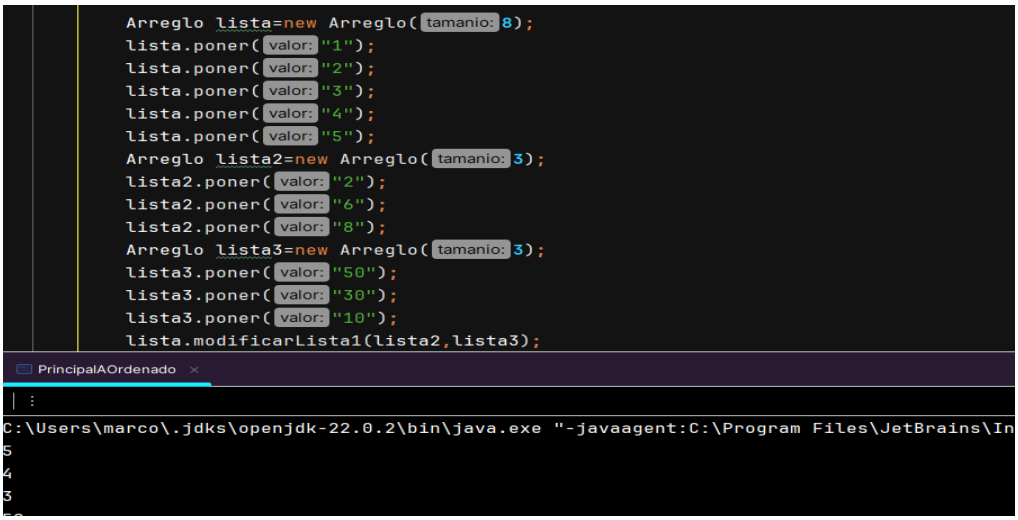


Imagen 19. Salida del método modificar lista

12. [declarado en interface ListaDatos] **public boolean retenerLista(ListaDatos lista2).** Debe dejar en la lista actual solo los elementos que se encuentran en lista2.

    La **visión General** es que vamos a vaciar nuestra lista 1 y la llenaremos con los datos de la lista 2.

    - Primero validamos que la lista2 sea una instancia de Arreglo.
    - Después hacemos el casting de lista 2 a Arreglo.
    - Validamos que en la lista 2 tenga la capacidad de almacenar los datos de la lista 1.
    - Vaciamos la lista1.
    - Hacemos un for para pasar los datos de la lista 2 a la lista 1.
    - Si todo sale bien retorna un true.

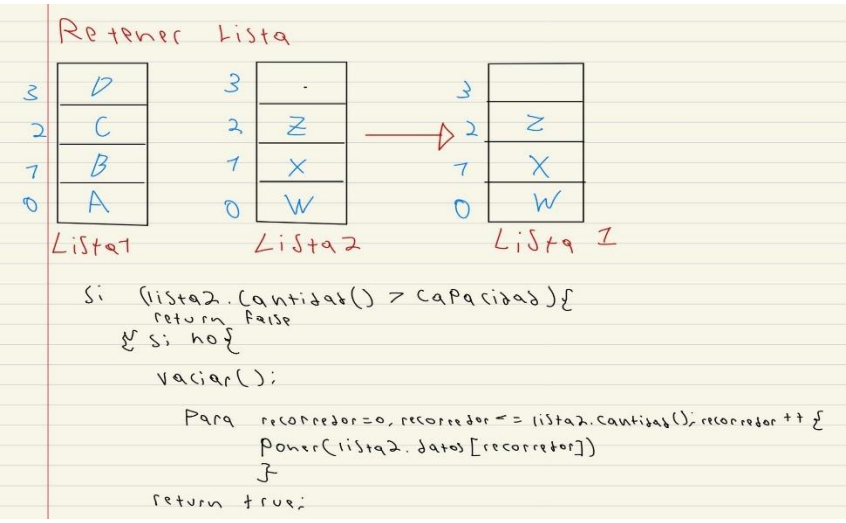


Imagen 20. Diseño del método retener lista

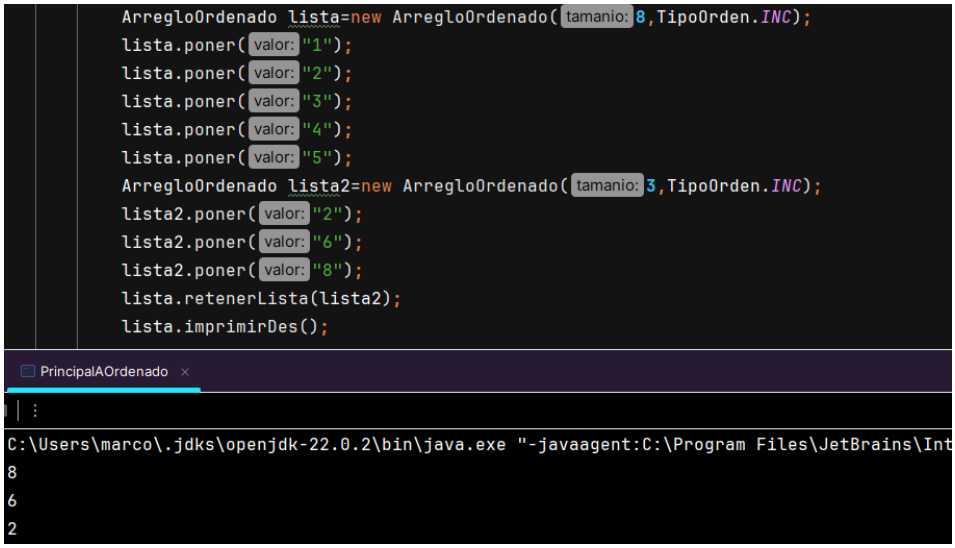


Imagen 21. Salida del método retener lista

Actividad 3

1. [declarado en interface ListaDatos] **public boolean poner(int indice, Object info).** Este método insertará en la posición "índice" el contenido de valor. o También modifique el método para sea sobre-escrito en la clase ArregloOrdenado de tal manera que se valide que siga ordenado.

   La **visión General** de este método es que en Arreglo se haga la inserción de manera normal en el índice y el objeto solicitado, pero en ArregloOrdenado primero se valida si en ese índice se sigue respetando el tipo de orden, si no se respeta regresa un false.

   ArregloOrdenado:

   - Primero se valida que el indice dado sea correcto y que el arreglo actual no este lleno.
   - Hacemos un if el cual separe que tipo de orden es.
   - Si es un tipo de orden INC lo que se realiza verificar que el valor a insertar sea mayor o igual que el dato anterior y menor o igual que el dato siguiente
   - SI es un tipo DEC va a verificar que el valor a insertar sea menor o igual que el dato anterior y mayor o igual que el dato siguiente
   - Hacemos un for para recorrer los datos del arreglo hacia la derecha/arriba y habrá un hueco para el nuevo valor a poner.
   - Pone el valor en su posición.
   - Se retorna un true si todo salió bien.

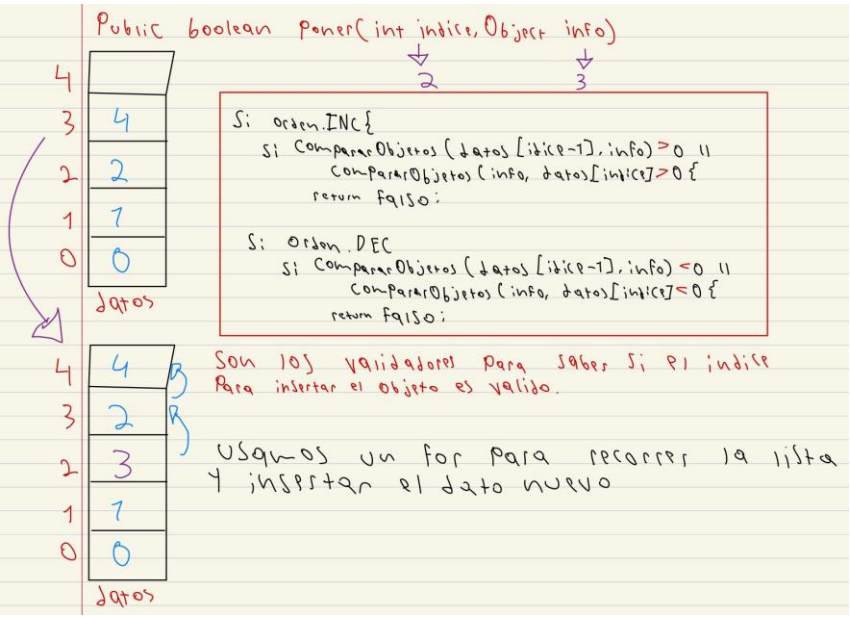


Imagen 22. Diseño del método poner

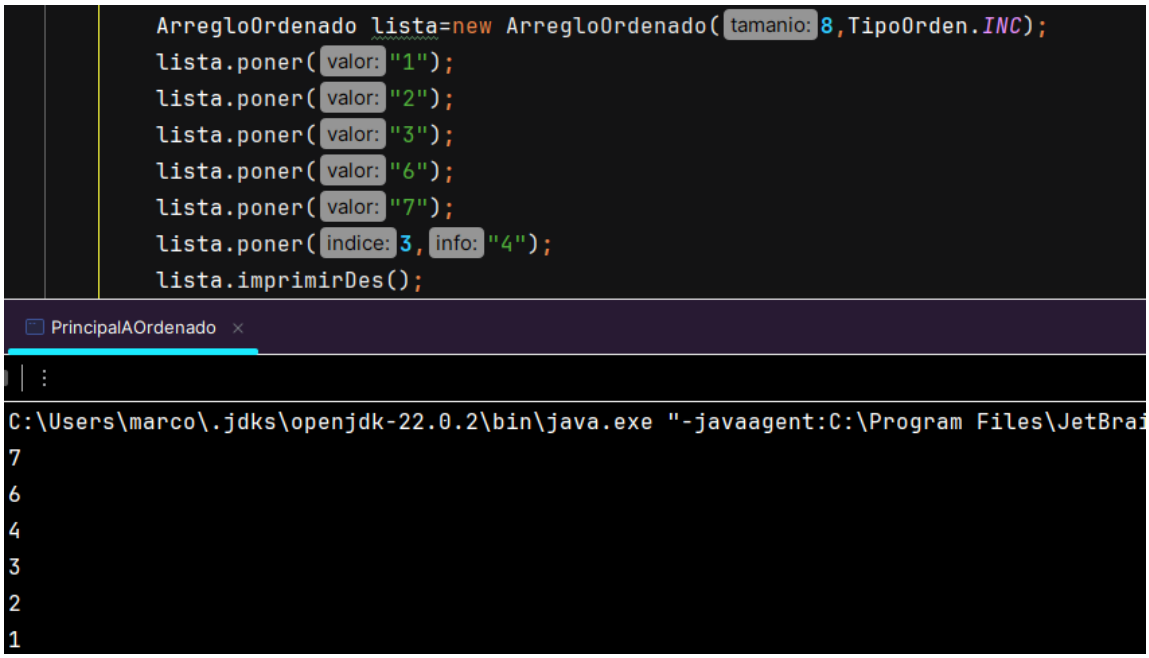


Imagen 23. Salida del método poner

2. [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 1. o 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.

   La visión General de este método es que se copia el contenido de la lista 2 a la actual, la lista actual se perderá ya que se remplazaran los datos de la lista 2.

   - Primero validamos que lista 2 sea un arregloOrdenado.
   - Transformamos de ListaDatos a Arreglo ordenado la lista2.
   - Validamos que la cantidad de la lista 2 excede la capacidad de la lista uno no podrá ingresar los datos y se retornará un false.
   - Verificaremos que se respete el mismo orden en los 2 arreglos.
   - Vaciamos la lista actual.
   - Con un for recorreremos la lista 2 y con el método poner insertaremos los datos de la tabla 2 a la tabla actual.

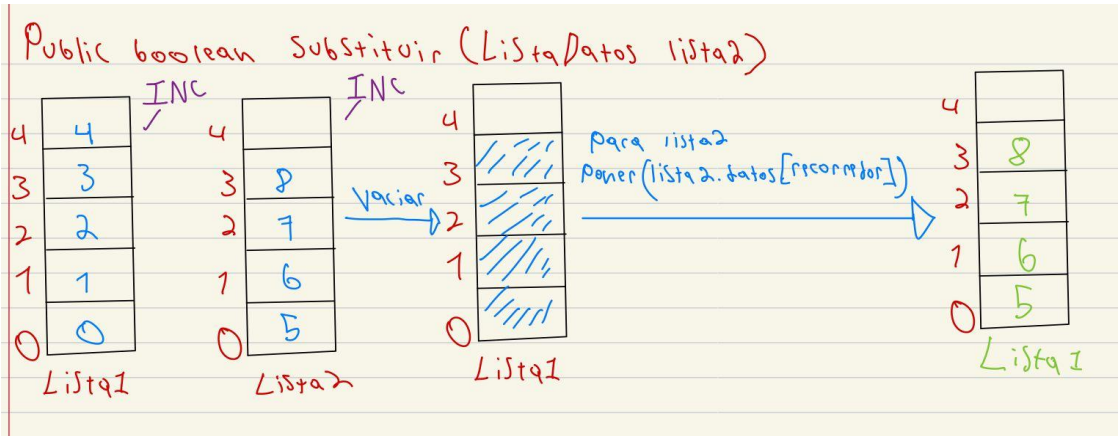


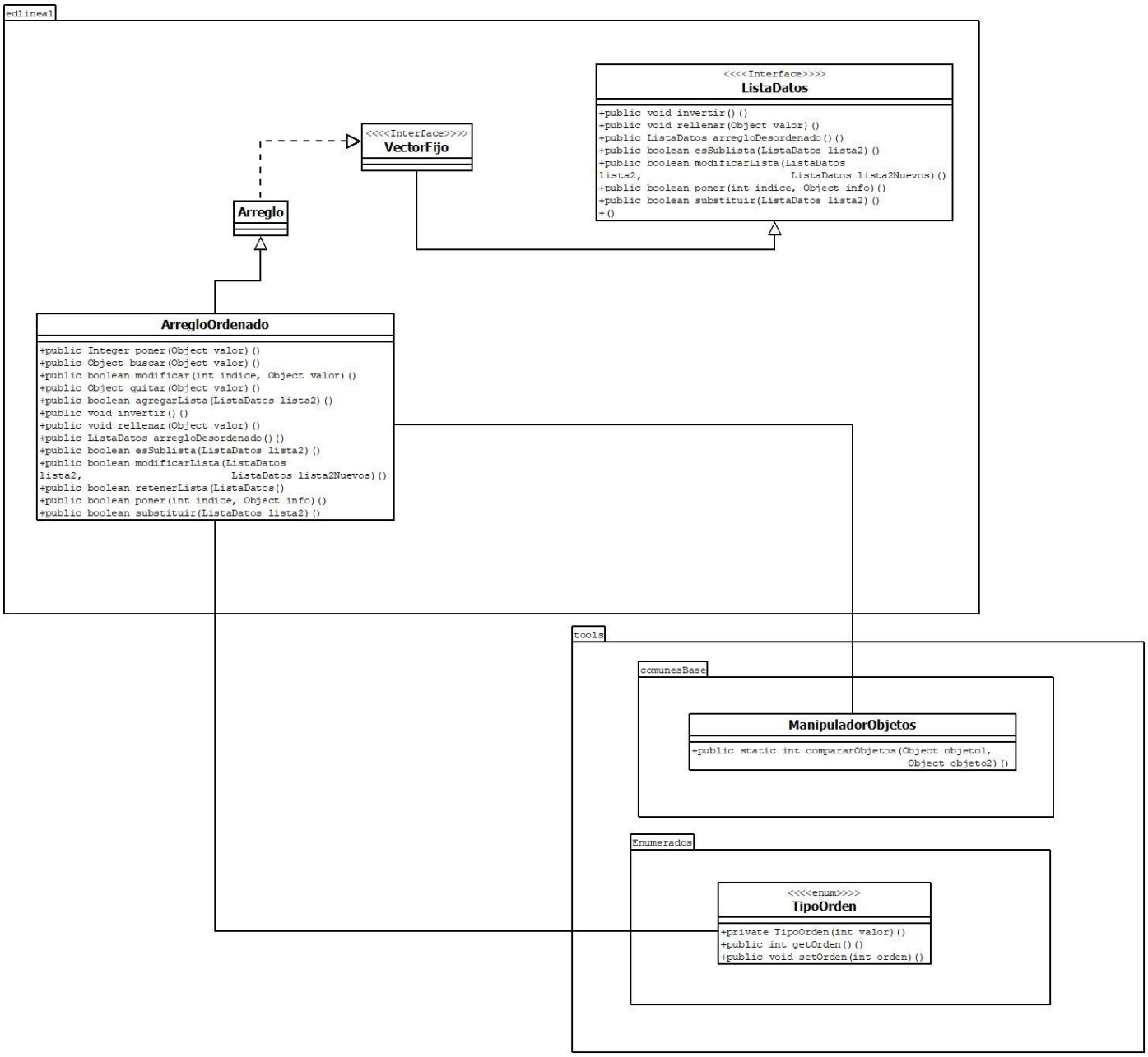
Imagen24. Diseño del método substituir

```
ArregloOrdenado lista=new ArregloOrdenado( tamanio: 8,TipoOrden.INC);
lista.poner( valor: "1");
lista.poner( valor: "2");
lista.poner( valor: "3");
lista.poner( valor: "6");
lista.poner( valor: "7");
ArregloOrdenado lista2=new ArregloOrdenado( tamanio: 3,TipoOrden.INC);
lista2.poner( valor: "2");
lista2.poner( valor: "6");
lista2.poner( valor: "8");
lista.substituir(lista2);
lista.imprimirDes();
```

```
PrincipalAOrdenado ×
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\Intelli
8
6
2
```

Imagen 25. Salida del método Substituir

Diagrama UML (Actividad 6)



Pre-Evaluación (Actividad 7)

#	Criterio	Respuesta
1	Cumple con la funcionalidad solicitada.	SI
2	Dispone de código autodocumentado.	SI
3	Dispone de código documentado a nivel de clase y método.	SI
4	Dispone de indentación correcta.	SI
5	Cumple la POO.	SI
6	Dispone de una forma fácil de utilizar el programa para el usuario.	SI
7	Dispone de un reporte con formato IDC.	SI
8	La información del reporte está libre de errores de ortografía.	SI
9	Se entregó en tiempo y forma la práctica.	NO
10	Incluye el código agregado en formato UML.	SI
11	Incluye las capturas de pantalla del programa funcionando.	SI
12	La práctica está totalmente realizada (especifique el porcentaje completado).	85% DE LA PRACTICA

Observaciones: No termine la actividad 4 por mal administración de mi tiempo, una disculpa.

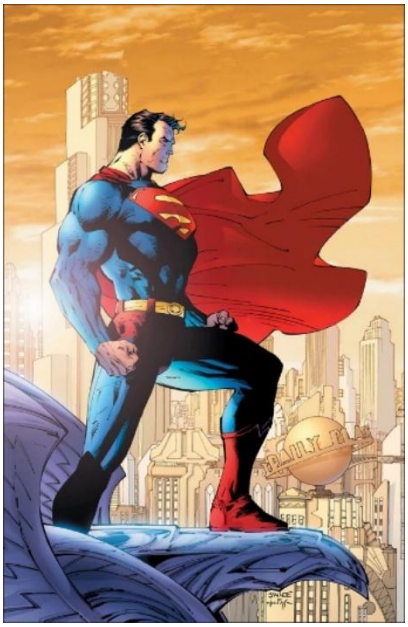


Imagen 26. El mejor super Héroe de la historia

Conclusión (Actividad Final)

Esta práctica fue épica ya que nos enseño la suma importancia de el buen manejo de ArreglosOrdenados y como se les podía sacar provecho a todas sus funcionalidades, dando sobre todo un gran énfasis al uso de arreglos unidimensionales.