Practica 3: Arreglos unidimensionales



Nombre: Jaime Gabriel Robles Félix

Grado y grupo: “4-B”

**Introducción**

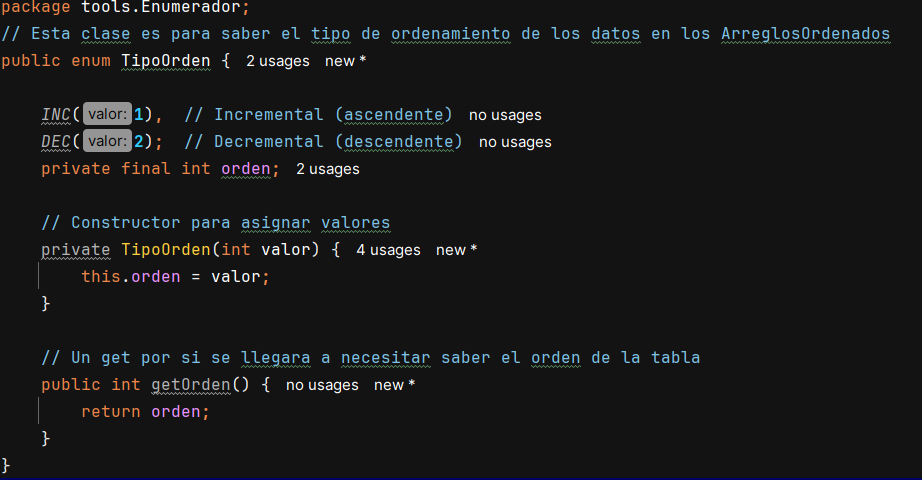
En esta práctica aremos mucho énfasis en los arreglos unidimensionales y ordenados, ya que al usar los métodos correctos se podría sacar mucho provecho a estas, estos ayudan a varias problemáticas de forma sencilla si se plantea bien el método y la lógica.

**Desarrollo**

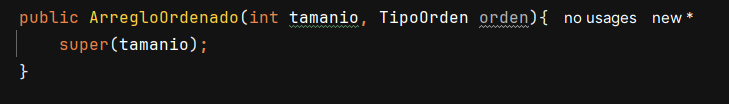
1. public ArregloOrden(int maximo, TipoOrdenamiento orden). Es el constructor del arreglo con orden. Este arreglo, además de indicar el tamaño, debe crear un enumerado en donde se indique el orden/acomodo de las inserciones, ya sea INC (1) o DEC (2) [incremental o decremental]. Ese orden se respetará 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 ASC, con una variable orden y un constructor el cual almacenara el atributo a la variable orden.

Este es el código autodocumentado para que lo entienda mejor el cual se explica cada parte de este:

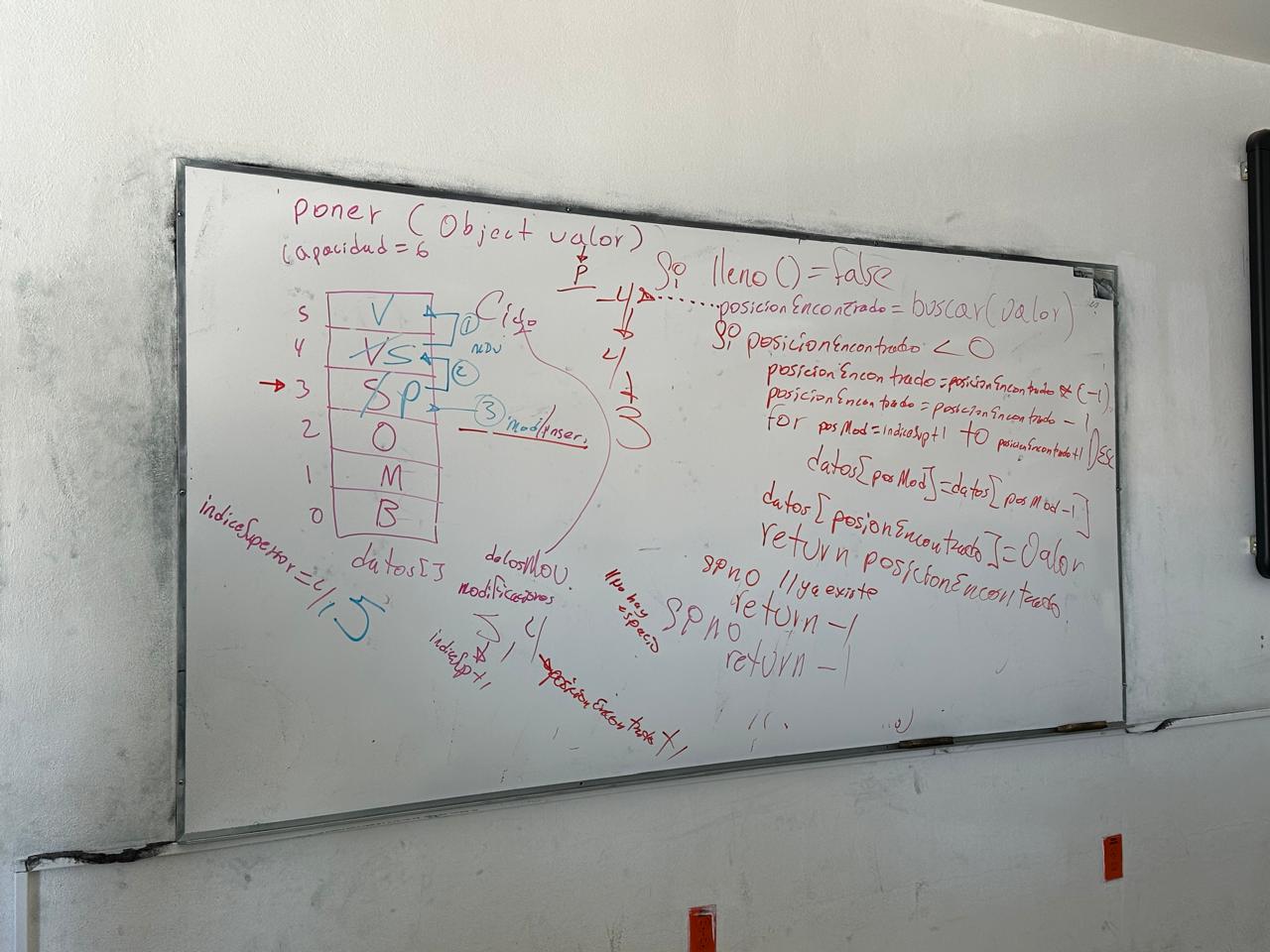


Este es nuestro nuevo constructor el cual recibe el TipoOrden.

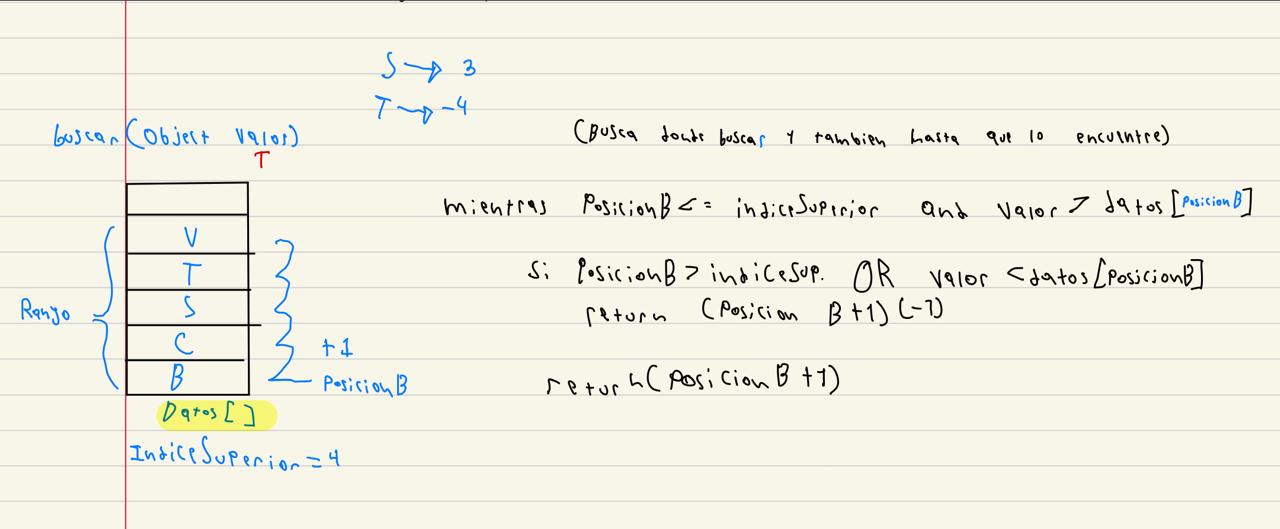


1. public Integer poner(Object valor). Insertar elementos mediante mecanismos ordenados en un orden definido en el constructor.

Este método lo realizamos en clase y este es el diagrama que realizo el profesor:



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

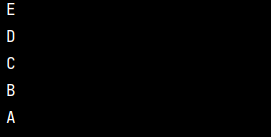
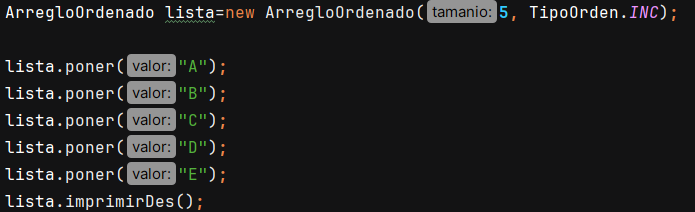
Este método también lo realizamos en clase de forma INCREMENTAL, este es el diseño:

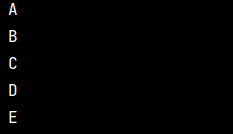
De forma resumida ya que se explico en clase, se busca si hay donde buscar y lo busca hasta que lo encuentre, si la tabla es **INCREMENTAL** se utiliza la herramienta de comparador de objetos la cual se compara los objetos y regresa el resultado en forma de numero positivo si el objeto 1 es mayor que 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.

En **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.



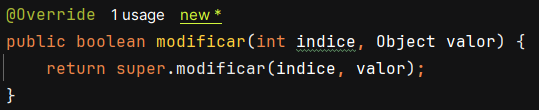
Se aprecia que el código es muy similar, únicamente cambia el signo en el “**while”** y en el “**if”** de cada uno, ya que el valor buscado es inverso.

Estos son los resultados utilizando cada tabla:



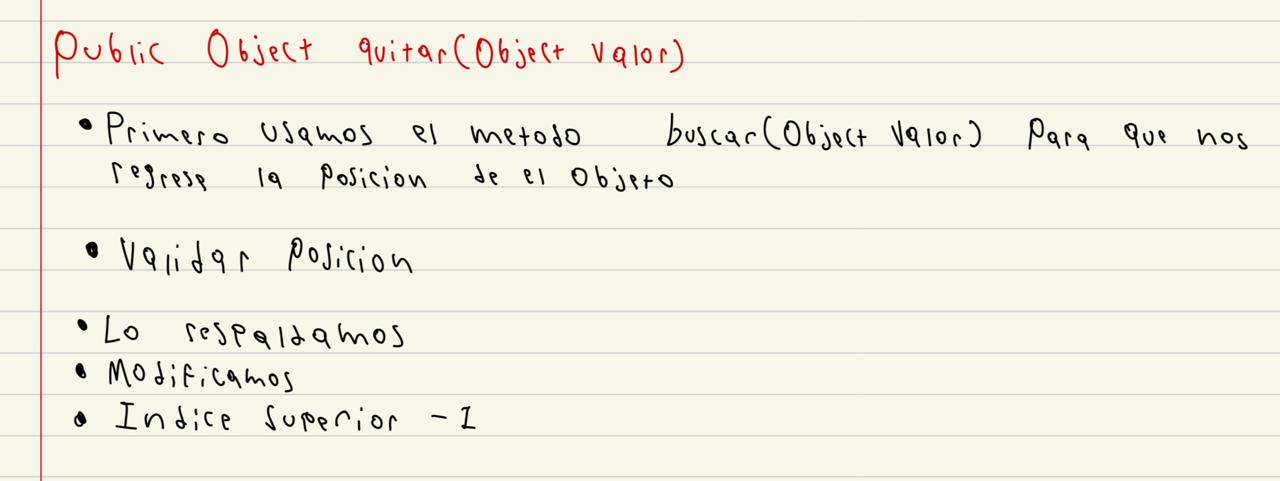
1. public boolean modificar(int indice, Object valor). Este método deberá sobrescribirse para que el contenido del arreglo se reorganice.

Únicamente se sobrescribió la función modificar tal cual como se pide:



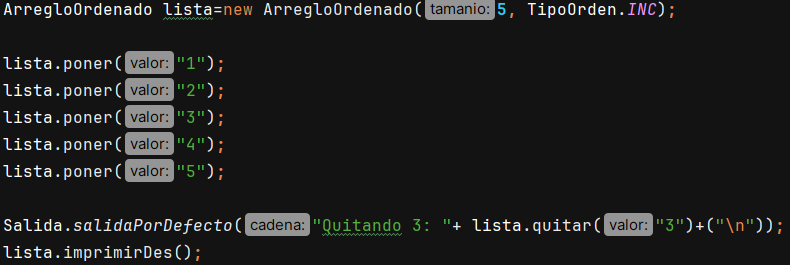
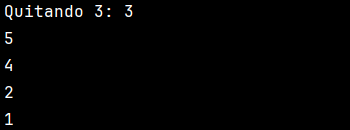
1. public Object quitar(Object valor). Eliminar un elemento del arreglo

Este es la idea y el diseño básico de mi método:

Ese es el proceso que debemos llevar para poder quitar un objeto en específico, para validar la posición utilizamos un método el cual valida que sea mayor a 0, cuando lo modificamos utilizamos un for el cual va a recorrer el arreglo remplazando el lugar del valor a quitar, por último, restamos -1 al índice superior.

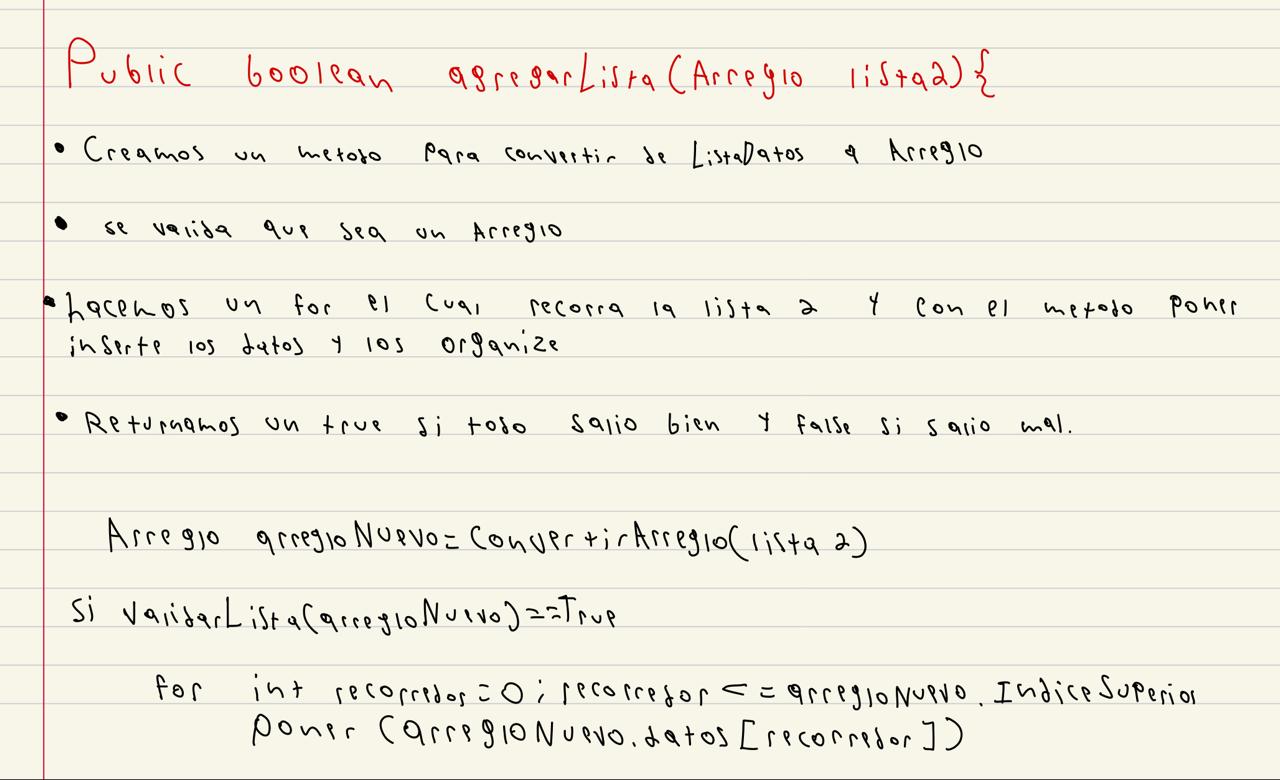


También realizamos un respaldo ya que es un protocolo de seguridad esencial el cual nunca puede faltar.

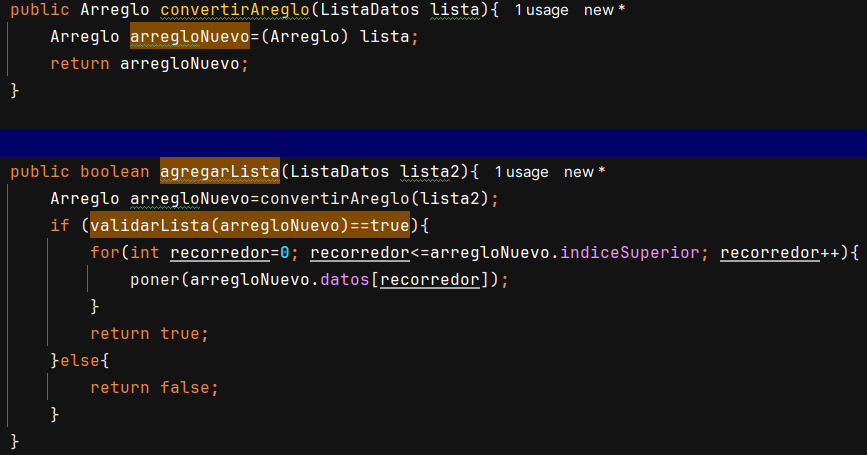


El 3 repetido después de quitando es el valor de respaldo el cual fue retornado, por ultimo la lista actualizada sin el valor que quitamos.

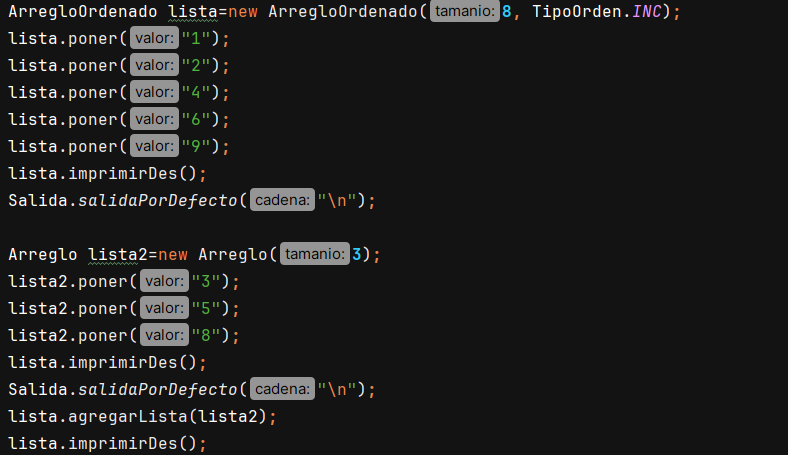
1. public boolean agregarLista(ListaDatos lista2). Debe permitir agregar los elementos de lista2 (que debe validar que sea un arreglo [ordenado o desordenado] en este caso) en el arreglo actual. Debe reorganizar todos los elementos insertados, de tal manera que el arreglo siga ordenado. Recuerde que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

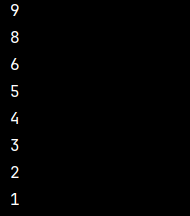
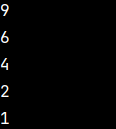
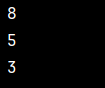


Análisis:



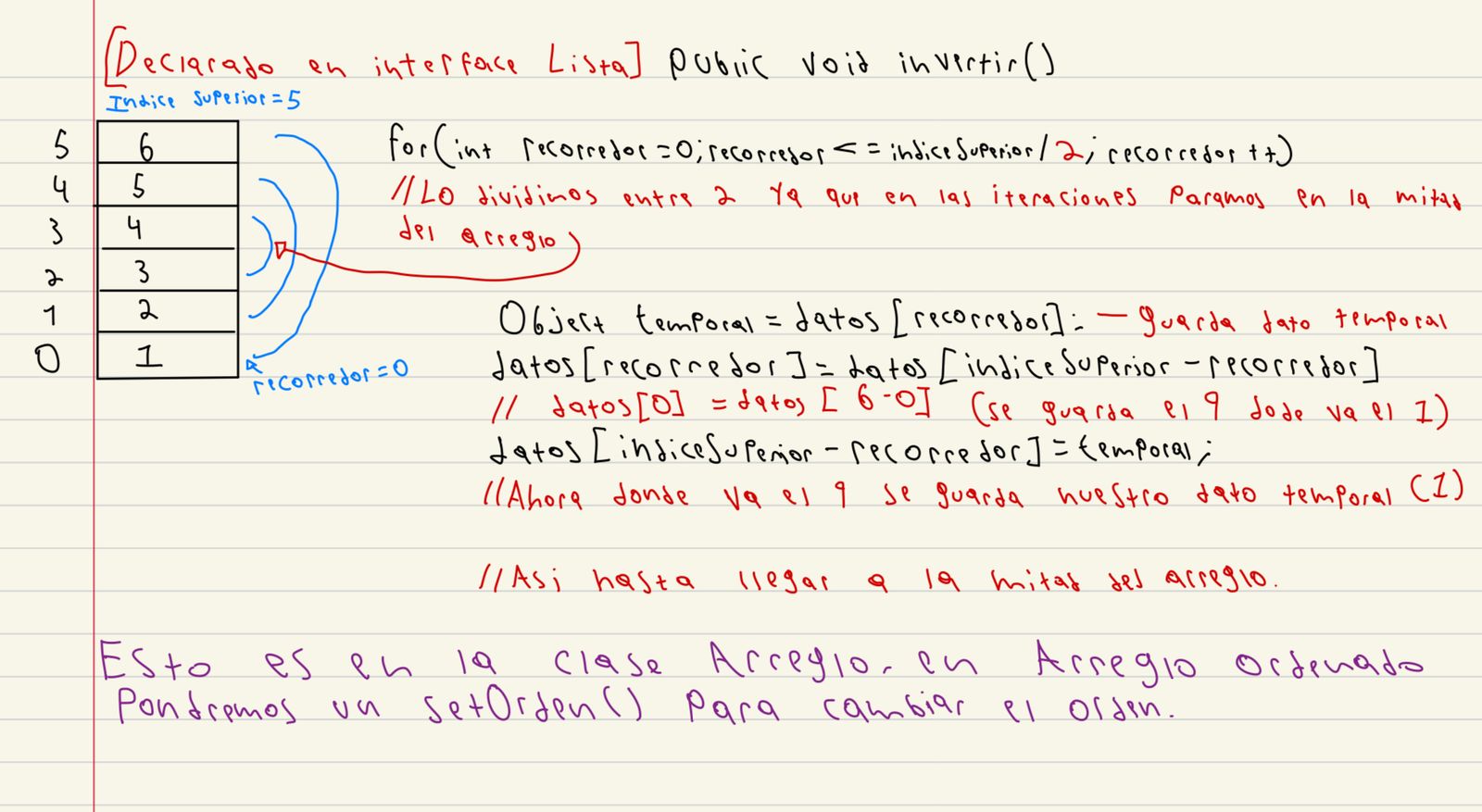
Este código fue muy simple ya que lo mas complicado que es ordenar ya lo hace nuestro método de poner.



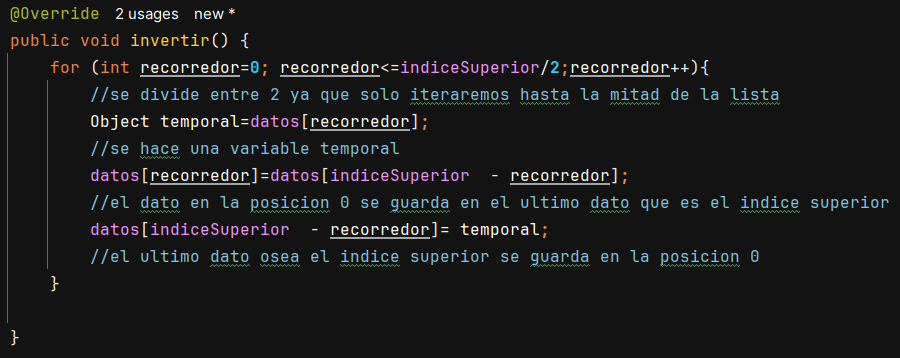


Ese es el resultado impreso en pantalla, la primera lista, la segunda lista y la primera lista con la funcion de agregarLista.

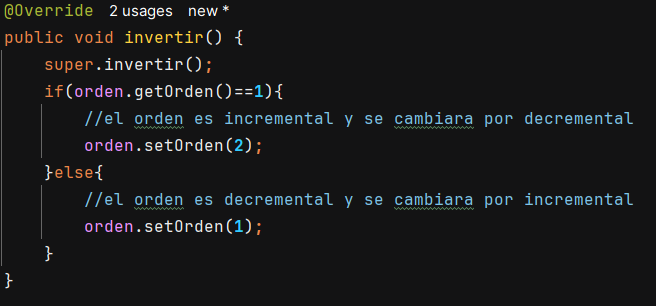
1. [declarado en interface ListaDatos] public void invertir(). Debe invertir el orden de los elementos del arreglo. Al mismo tiempo que se cambia el orden, debe cambiarse el valor de la variable TipoOrdenamiento, sino mostrará inconsistencias.

Análisis:

Este es mi análisis completo de como invertir el arreglo, es una forma sencilla pero si me costo algo de tiempo descifrar la manera correcta de hacerlo.



Este es el código en la clase **Arreglo** el cual se diseñó en el análisis.



Este es en la clase ArregloOrdenado el cual lo único que hace es que hereda el método invertir y únicamente realiza los cambios del oren con un setOrden.