Practica 3: Arreglos unidimensionales

Nombre: Jaime Gabriel Robles Félix

Grado y grupo: “4-B”

Indice

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.
   `

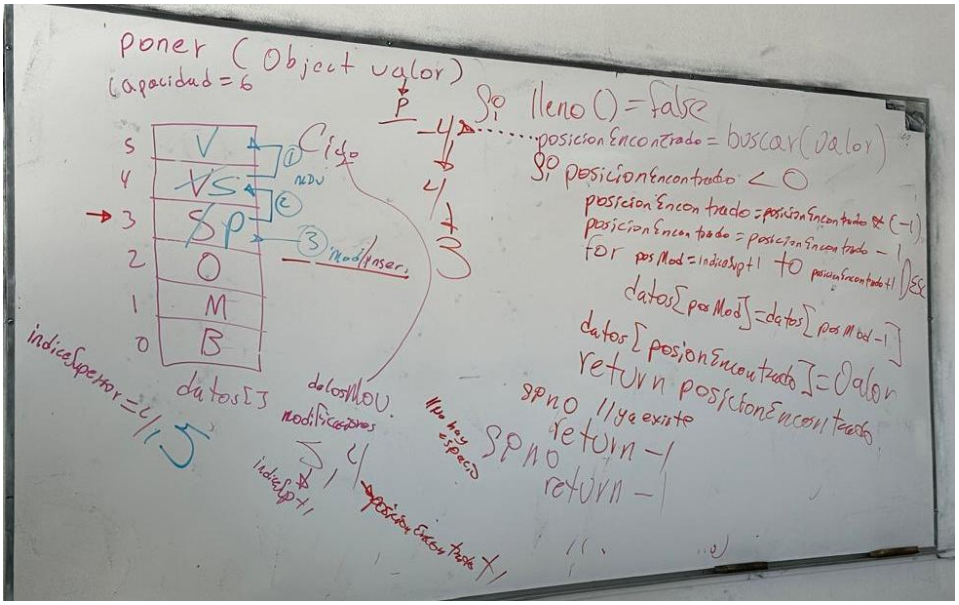


Imagen 2. Análisis y diseño en clase

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

   Este método también lo realizamos en clase de forma INCREMENTAL.

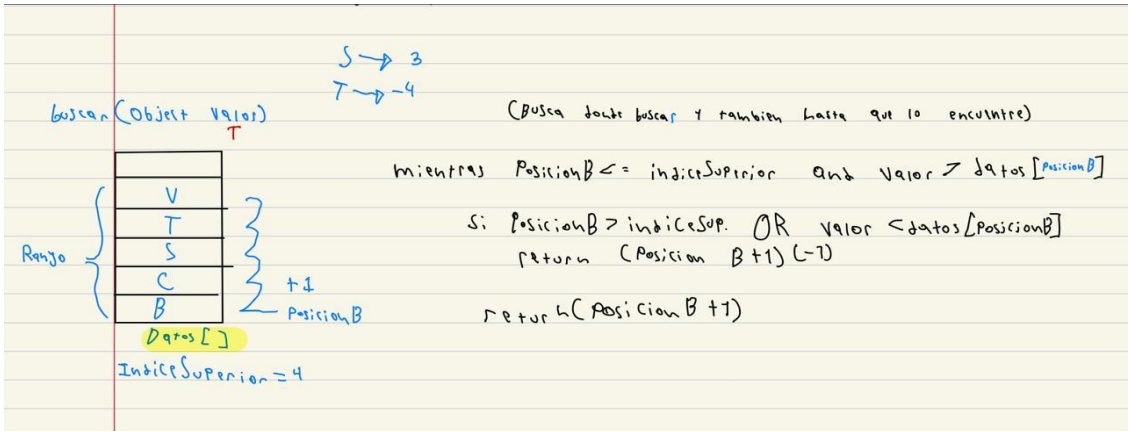


Imagen 3. Diseño de la forma incremental

De forma resumida ya que se este diseño se realizó 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.

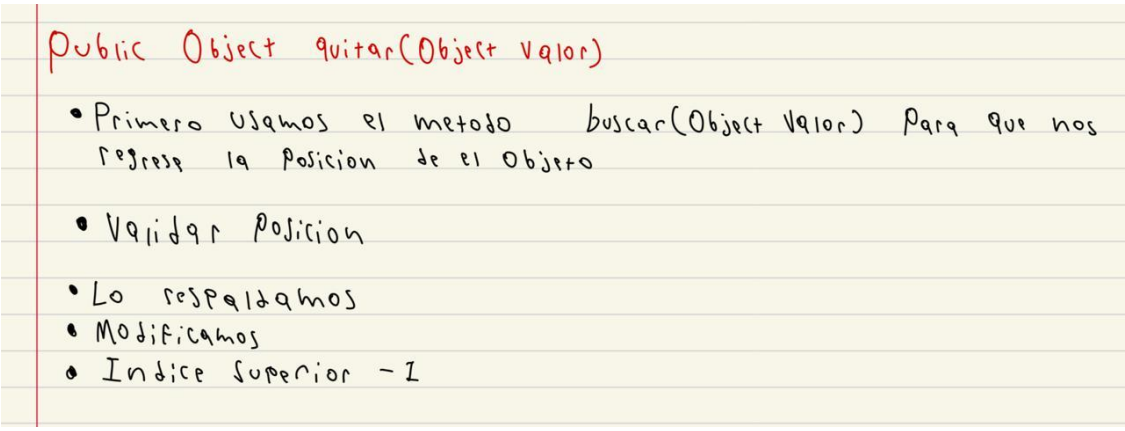
Estos son los resultados utilizando cada tabla:

4. 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:

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

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



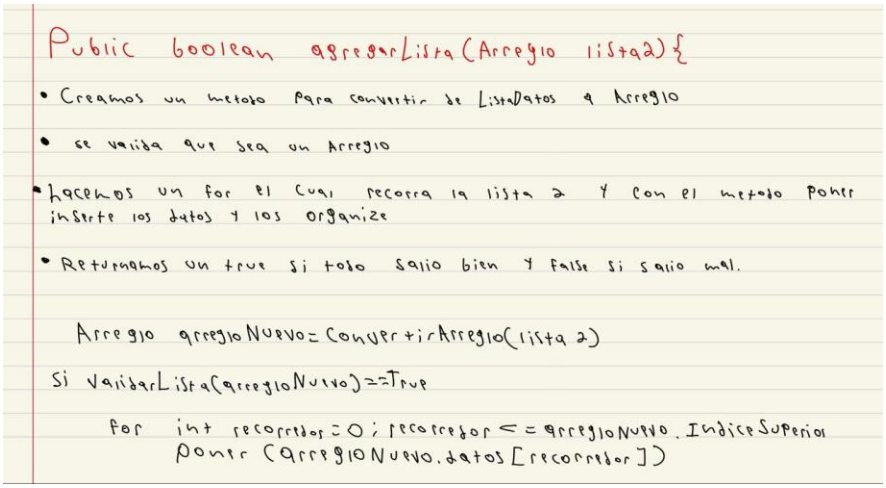
   Ese es el proceso que debemos llevar para poder quitar un objeto en específico, para validar la posición utilizamos un método el cual valida que sea mayor a 0, cuando lo modificamos utilizamos un for el cual va a recorrer el arreglo remplazando el lugar del valor a quitar, por último, restamos-1 al índice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

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. CORREGIR

Análisis:

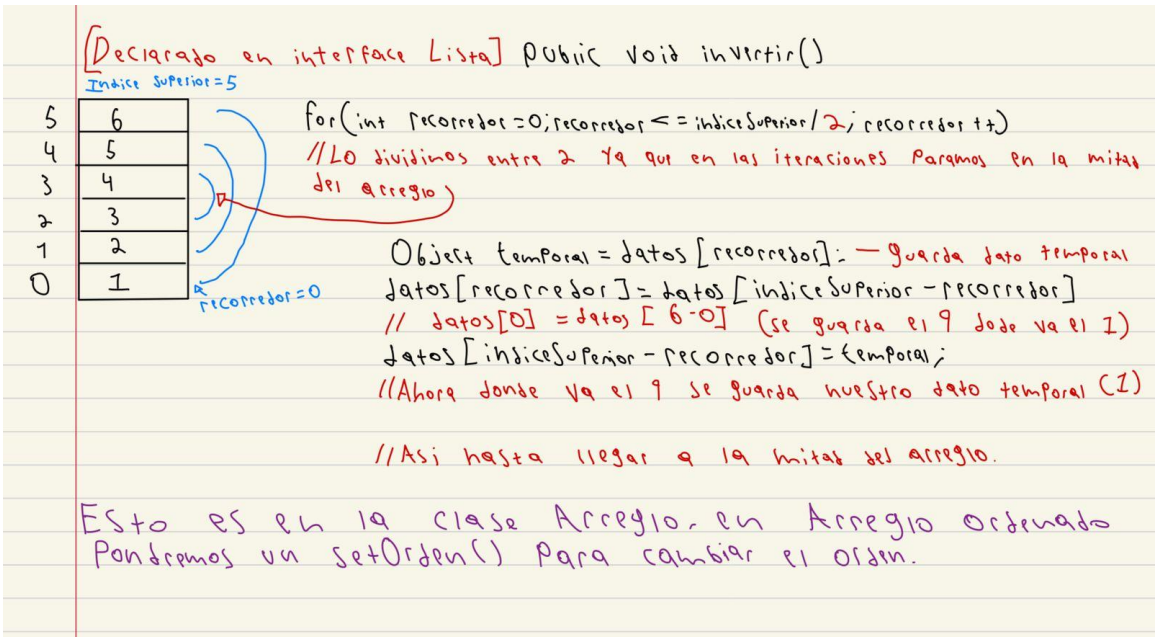


Lista2 se convierte en un ArregloOrdenado, código fue muy simple ya que lo mas complicado que es ordenar ya lo hace nuestro método de poner.

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

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.

   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.

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.

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.

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 lista2Nuebos. 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.

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

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.

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.