Practica 2

Nombre: Jaime Gabriel Robles Félix

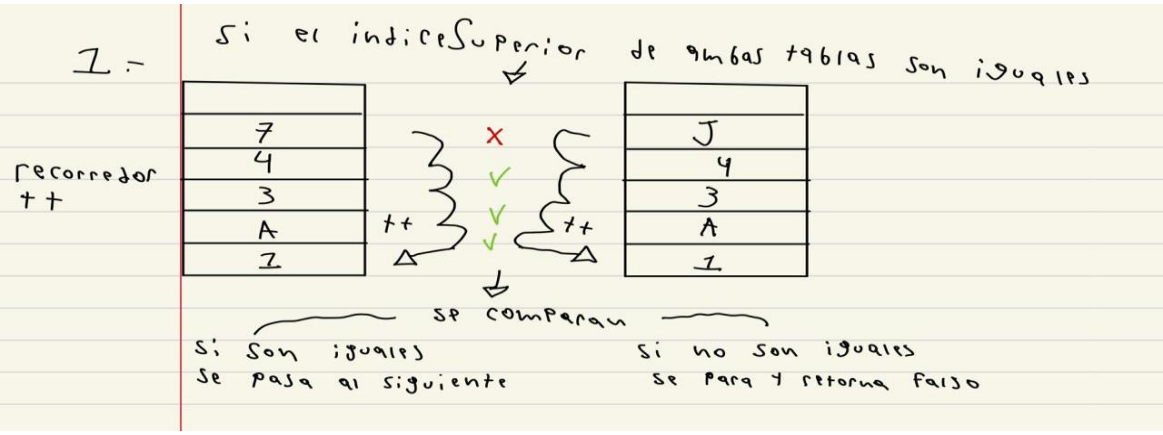
Grado y grupo: “4-B”

Introducción

- En esta practica estaremos trabajando mucho con arreglos y métodos para estos, el manejo de arreglos puede parecer sencillo, pero si requiere de mucha deducción y razonamiento.

Desarrollo

1. Lo que realice es que se le pasa al método una lista la cual se compara con la lista actual para descifrar si son iguales, este es mi diseño:




```java
//se validara si 2 listas son iguales
public boolean esIgual(Arreglo lista){  1 usage
    //primero se valida la lista
    validarLista(lista);
    //asigno los valores a comparar
    Object comparador1;
    Object comparador2;
    //este if es si por logica uno es de un tamaño diferente que el otro, es falso
    if (indiceSuperior==lista.indiceSuperior){
        for (int recorredor=0; recorredor<=indiceSuperior; recorredor++){
            comparador1=datos[recorredor];
            comparador2=lista.datos[recorredor];
            if (comparador1.toString().equals(comparador2.toString())==false){
                return false;
            }
        }return true;
    }else{
        return false;
    }
}
```

```
Run  principalArreglo
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ I
Son iguales
```

Al for le damos el límite del indiceSuperior ya que no queremos buscar en donde no hay datos, después cuando entramos al for se asignan los comparadores, se hace un if el cual convierte a los dos en .toString y los compara, si son iguales se mantiene en el for y si son diferentes arroja un false, en caso de que se cumpla que todos los datos son iguales retornara un true.

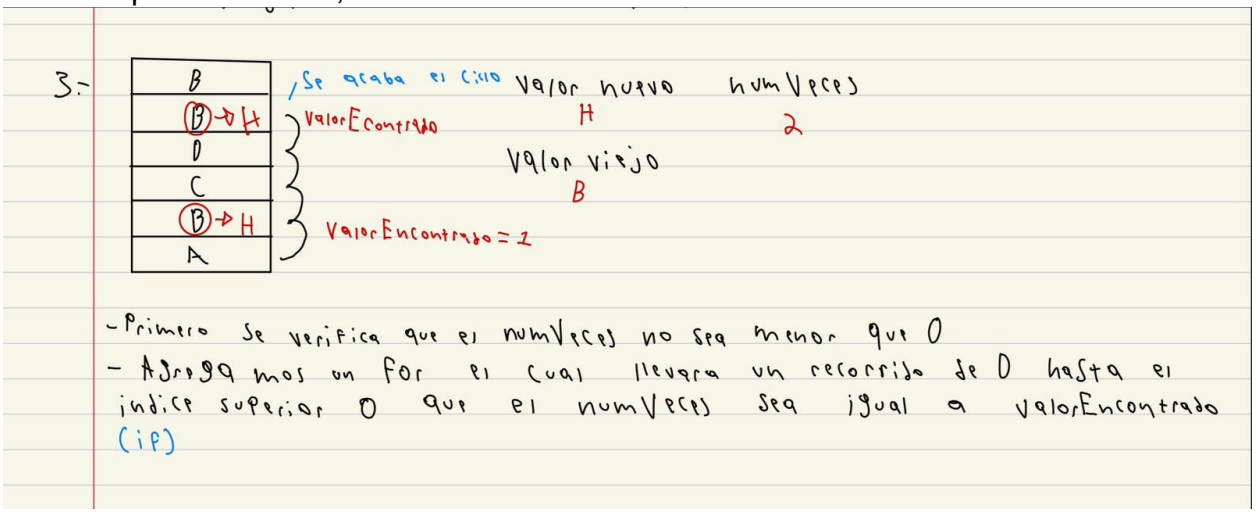
2. Regresar el objeto de la posición índice:

```
    //Este metodo obtiene un objeto del arreglo indicado por la posicion del indice
    public Object obtener(int indice) {  no usages
        if(validaPosicion(indice)==true){
            return datos[indice];
        }else{
            return null;
        }
    }
```

```
principalArreglo ×
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\Intel
Valor obtenido en el indice 2: M
```

Como se muestra es un código muy básico el cual utilizamos un simple if para validar la posición y si cumple retorna el objeto.

3. modifica el elemento viejo por el elemento nuevo haciendo una búsqueda y modificación del número de veces de ocurrencias del elemento encontrado indicado por numVeces, mi diseño:



Este es el código main y el resultado en terminal:

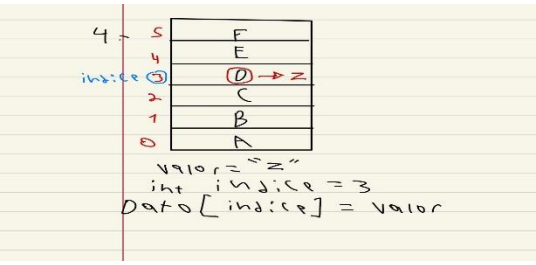
```java
public class principalArreglo {
    public static void main(String[] args) {
        Arreglo lista = new Arreglo( tamanio: 5);
        lista.poner( valor: "S");
        lista.poner( valor: "T");
        lista.poner( valor: "M");
        lista.poner( valor: "S");
        lista.poner( valor: "N");
        lista.poner( valor: "R");

        lista.modificar( valorAntiguo: "S",  valorNuevo: "H",  numVeces: 1);
        lista.imprimir();
```

principalArreglo

```
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program
H

T

M

S

N
```

Como vemos en la imagen únicamente remplazo el primer valor antiguo encontrado por el número de veces que le dimos, si le agregamos 2 si cambiarían los 2 valores antiguos.

4. Diseño:



```java
public class principalArreglo {
    public static void main(String[] args) {
        Arreglo lista = new Arreglo( tamanio: 5);
        lista.poner( valor: "S");
        lista.poner( valor: "T");
        lista.poner( valor: "M");
        lista.poner( valor: "S");
        lista.poner( valor: "N");
        lista.poner( valor: "R");
        lista.modificar( indice: 2, valor: "Z");
        lista.imprimir();
    }
}
```

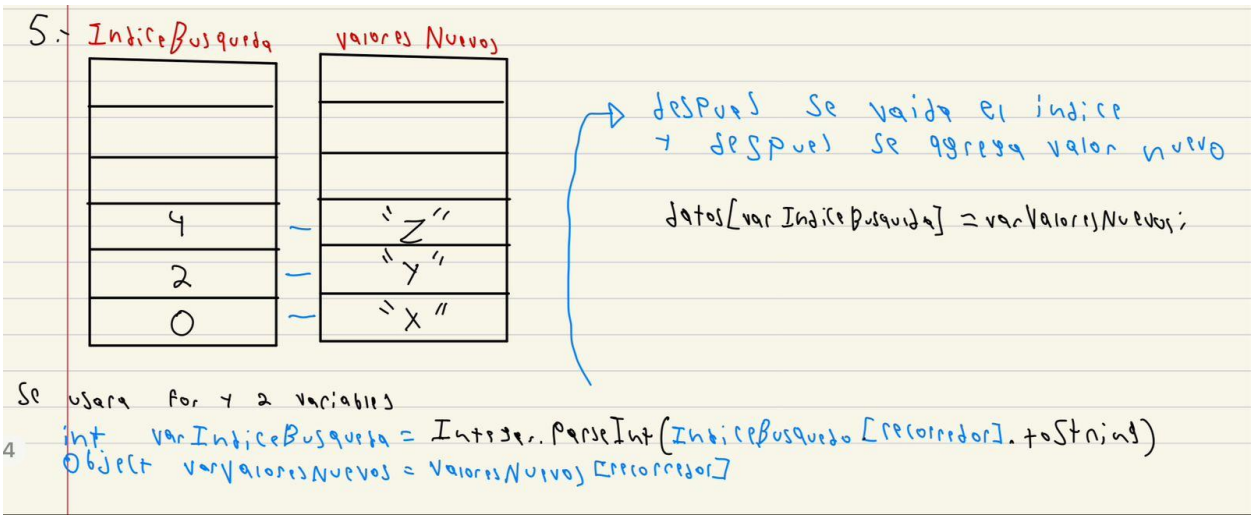
```
principalArreglo
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Pro
S

T

Z

S

N
```

El valor que teníamos en el índice 2 (“M”) lo remplazamos por el valor (“Z”)

```java
public boolean modificar(int indice, Object valor){  1 usage
    if (validaPosicion(indice)==true){
        return false;
    }
    datos[indice]=valor;

    return true;
}
```

El código fue muy simple ya que solo usamos un if para la validación del índice y la que asigna el nuevo valor.

5. Así fue el diseño de este método:



El resultado de mi programa:

```
public static void main(String[] args) {
    lista.poner( valor: "M");
    lista.poner( valor: "S");
    lista.poner( valor: "N");
    lista.poner( valor: "R");

    Arreglo indiceBusqueda= new Arreglo( tamanio: 3);
    Arreglo valorNuevo= new Arreglo( tamanio: 4);

    indiceBusqueda.poner( valor: 0);
    indiceBusqueda.poner( valor: 2);
    indiceBusqueda.poner( valor: 4);

    valorNuevo.poner( valor: "X");
    valorNuevo.poner( valor: "Y");
    valorNuevo.poner( valor: "Z");

    lista.modificarLista(indiceBusqueda,valorNuevo);
    lista.imprimir();
```

principalArreglo

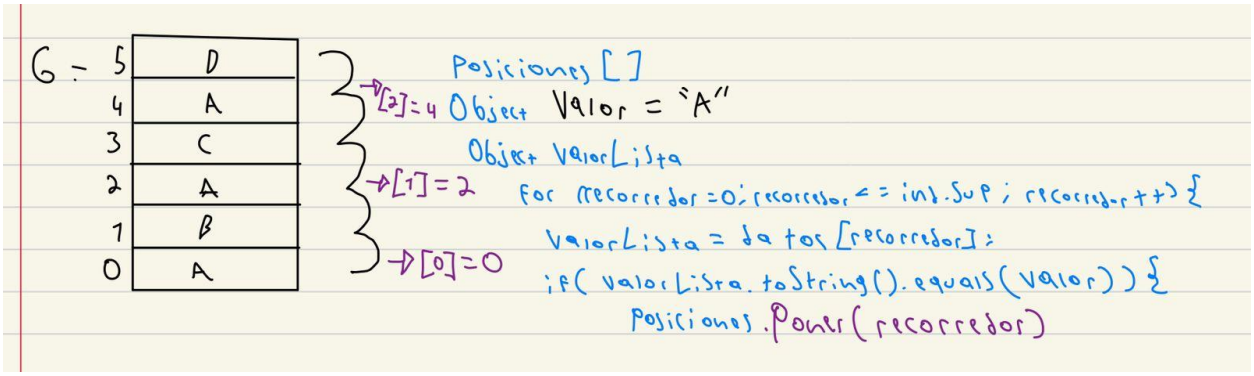
```
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\
X

T

Y

S

Z
```

Se le pasaron 2 arreglos al método los cuales son los que se muestran en la imagen, ahí se ve el resultado obtenido, aquí está el método completo:

```java
public boolean modificarLista(Arreglo indicesBusqueda, Arreglo valoresNuevos) {  no usages
    int variableIndiceBusqueda;
    Object variableValorNuevo;
    if (indicesBusqueda.indiceSuperior==valoresNuevos.indiceSuperior){
        for (int recorredor=0; recorredor<=indicesBusqueda.indiceSuperior; recorredor++){
            variableIndiceBusqueda=Integer.parseInt(indicesBusqueda.datos[recorredor].toString());
            variableValorNuevo=valoresNuevos.datos[recorredor];

            if (validaPosicion(variableIndiceBusqueda)==true){
                datos[variableIndiceBusqueda]=variableValorNuevo;
            }
        }return true;
    }else{
        return false;
    }
```

Se hacen 2 variables para guardar el contenido de las listas, después se hace una validación ya que si una lista tiene un índice superior de 3 y el otro de 5 regresa un falso ya que no concuerdan entre ellas y algún dato quedaría null, adentro del for se hace la asignación de las variables a sus valores correspondientes, por ultimo se valida si el valor extraído de la tabla índice búsqueda esta en el rango de nuestra lista a modificar y se realiza la modificación correspondiente.

6. Diseño:



```java
public class principalArreglo {
    public static void main(String[] args) {
        Arreglo lista = new Arreglo( tamanio: 6);
        lista.poner( valor: "S");
        lista.poner( valor: "T");
        lista.poner( valor: "S");
        lista.poner( valor: "N");
        lista.poner( valor: "S");
        lista.poner( valor: "P");


        Arreglo posiciones=lista.buscarValores("S");
        posiciones.imprimir();
```

principalArreglo

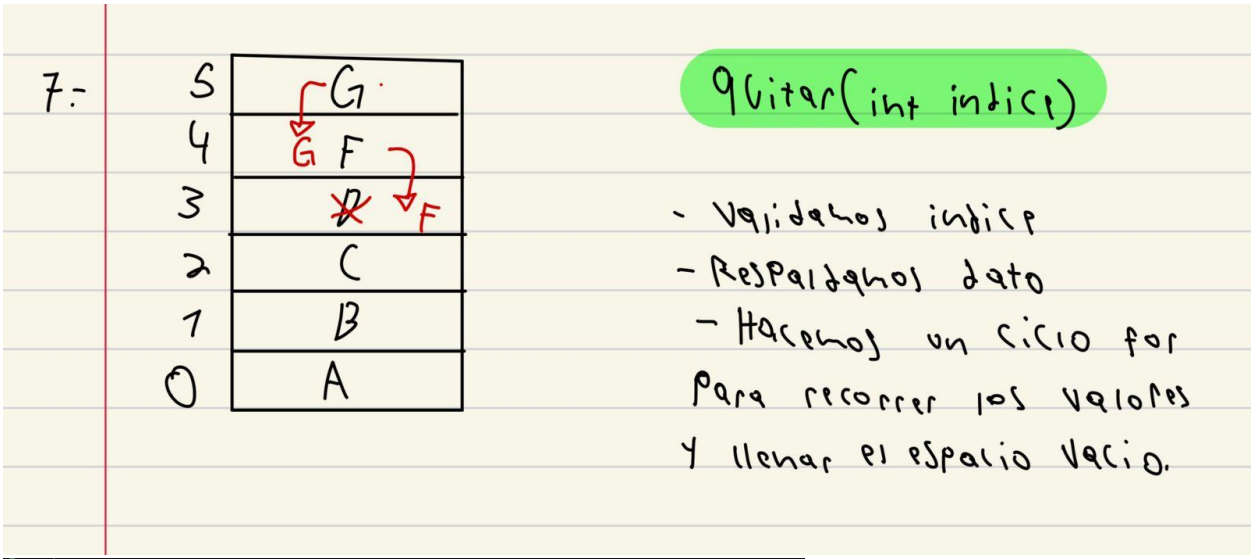
```
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program F
0

2

4
```

Este es el resultado, el Arreglo posiciones lo agregué ya que el método retornaba un Arreglo y posteriormente lo imprimí

```java
public Arreglo buscarValores(Object valor) {  1 usage
    Object valorLista;
    Arreglo posiciones;
    posiciones=new Arreglo( tamanio: 10);
    for (int recorrido=0; recorrido<=this.indiceSuperior;recorrido++){
        valorLista=datos[recorrido];
        if (valorLista.toString().equals(valor.toString())){
            posiciones.poner(recorrido);
        }
    }
    return posiciones;
}
```

Se crea el arreglo de las posiciones y creamos una variable para ayudarnos a hacer una comparación, se crea un for el cual hacemos la asignación de la variable anterior y la comparamos para ver si el valor es el mismo, si se cumple esta posición, se guarda en la lista posiciones el índice de donde se encontró la similitud de los valores.

7. Diseño:




```java
public class principalArreglo {
    public static void main(String[] args) {
        Arreglo lista = new Arreglo( tamanio: 6);
        lista.poner( valor: "S");
        lista.poner( valor: "T");
        lista.poner( valor: "S");
        lista.poner( valor: "N");
        lista.poner( valor: "S");
        lista.poner( valor: "P");

        lista.quitar( indice: 3);
        lista.imprimir();
```

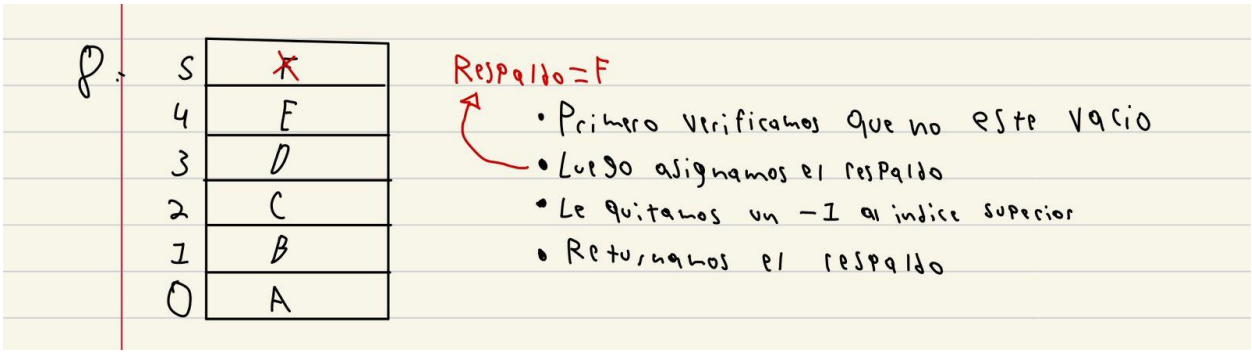
```
principalArreglo
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\
S

T

S

S

P
```

El resultado obtenido fue la eliminación del índice, y la redistribución de la lista a la izquierda.

```java
public Object quitar(int indice){  no usages
    Object respaldo;
    if(validaPosicion(indice)==true){
        respaldo=datos[indice];
        for(int modificacion=indice; (modificacion<=indiceSuperior-1);modificacion++){
            datos[modificacion]=datos[modificacion+1];
        }
        indiceSuperior=indiceSuperior-1;
        return respaldo;
    }else{
        return null;
    }
```

Primero creamos una variable de respaldo POR SEGURIDAD, después validamos el índice, asignamos el valor del respaldo y por ultimo redistribuimos la lista para llenar el espacio vacío generado.

8. Diseño:



Resultado:

```java
public class principalArreglo {
    public static void main(String[] args) {
        Arreglo lista = new Arreglo( tamanio: 6);
        lista.poner( valor: "S");
        lista.poner( valor: "T");
        lista.poner( valor: "S");
        lista.poner( valor: "N");
        lista.poner( valor: "S");
        lista.poner( valor: "P");

        lista.quitar();
        lista.imprimir();
```

```
principalArreglo
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-
S

T

S

N

S
```

Método:

```
public Object quitar(){  no usages
    if(vacia()==false){
        Object datoExtraido=datos[indiceSuperior];
        indiceSuperior= indiceSuperior - 1;
        return datoExtraido;
    }else{
        return null;
    }
}
```

9. Resultado:

```
public class principalArreglo {
    public static void main(String[] args) {
        Arreglo lista = new Arreglo( tamanio: 6);
        lista.poner( valor: "S");
        lista.poner( valor: "T");
        lista.poner( valor: "S");
        lista.poner( valor: "N");
        lista.poner( valor: "S");
        lista.poner( valor: "P");

        lista.vaciar();
        lista.imprimir();
```

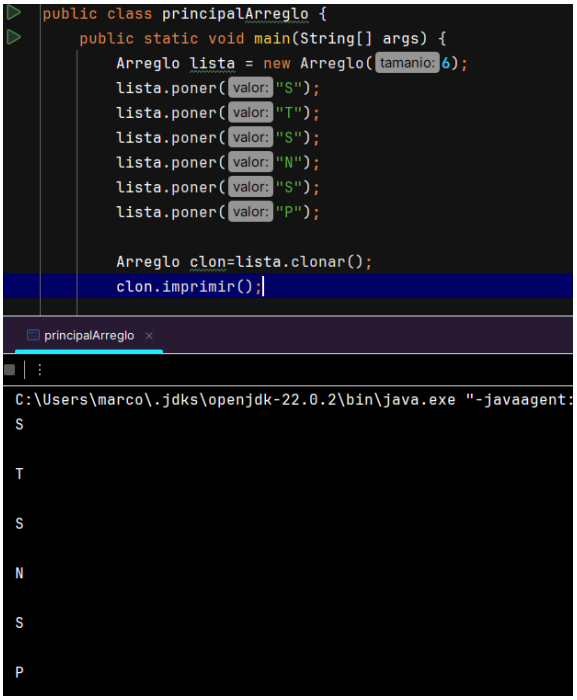
```
principalArreglo ×
:
C:\Users\marco\.jdks\openjdk-22.0.2\bin\java.exe "-
```

Método:

```java
public void vaciar() {  1 usage
    indiceSuperior = -1;
}
```

Se restableció el índice superior para que se considerara el arreglo vacío, recuerden que los datos siguen estando ahí, pero son ignorados, únicamente se pueden sobrescribir para eliminarlos.

10. Resultado:

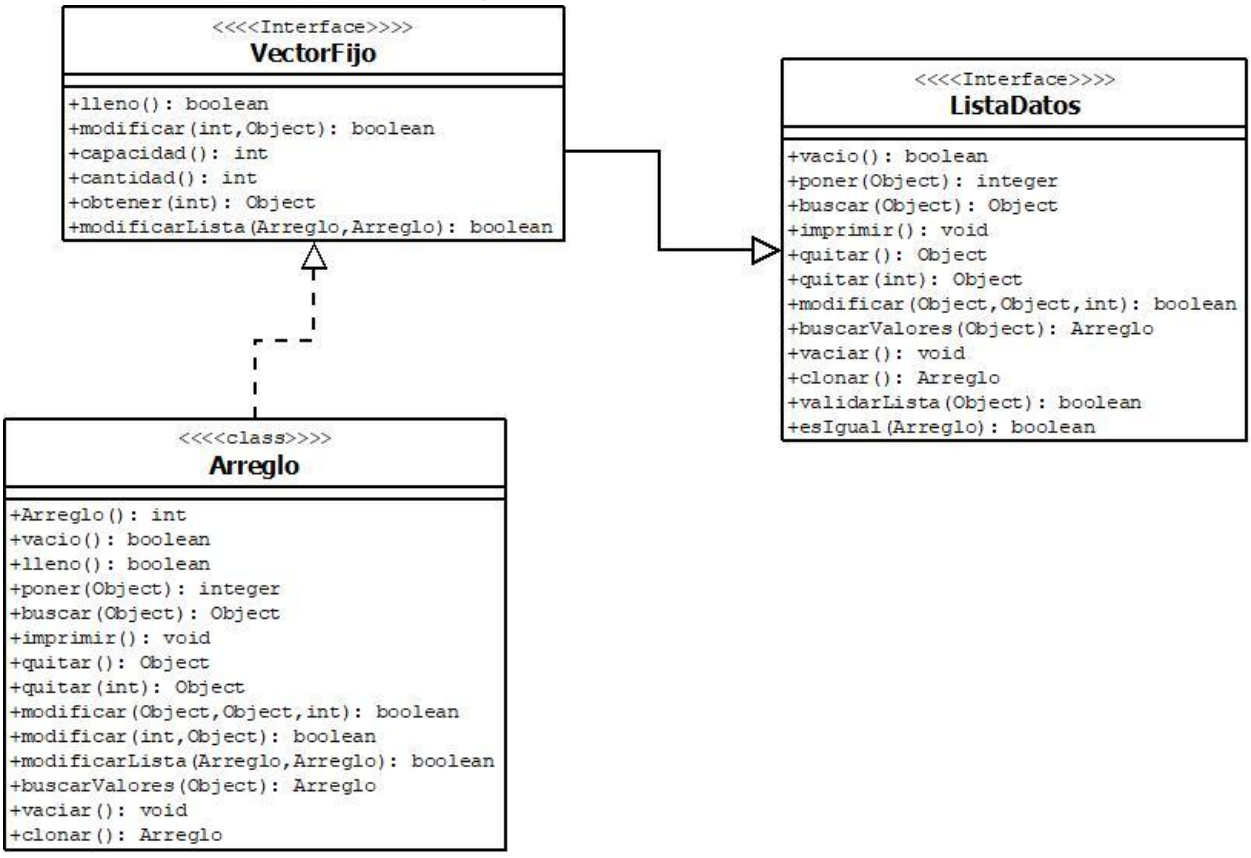


Método:

```java
public Arreglo clonar() {  1 usage
    Arreglo clon = new Arreglo(capacidad);

    for (int recorredor = 0; recorredor <= indiceSuperior; recorredor++) {
        clon.datos[recorredor] = datos[recorredor];
    }

    clon.indiceSuperior = indiceSuperior;

    return clon;
```

Primero creamos nuestro nuevo arreglo el cual será el clon, luego usamos un for el cual nos va a clonar los datos de cada lista, por último, ajustamos el parámetro de índice superior, retornamos el arreglo clonado;

Diagrama UML



Pre-evaluación para prácticas de Laboratorio de Estructuras de Datos
pre-evaluación del alumno

1. Cumple con la funcionalidad solicitada. **SI**
2. Dispone de código auto-documentado. **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).
65%

Observaciones (opcional): Al ser mi primera practica en la materia me confíe y el tiempo paso factura, se requiere de mucha concentración y análisis para desarrollar las practicas.

Conclusión

- Esta practica fue muy importante ya que nos enfocamos en el uso de arreglos los cuales son primordiales para la estructura de datos, cada nuevo método pedido fue un reto el analizarlo, comprenderlo y poder resolverlo aplicando conocimientos anteriores.