Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software.

Nombre de la Práctica Listas doblemente ligadas.

Numero de Práctica 19

Nombre de la carrera Ingeniería de Software

Nombre de la materia Lab. Estructuras de Datos

Nombre del alumno Jesús Manuel Juárez Pasillas

Nombre del docente Aldonso Becerra Sánchez

Fecha: 06/10/2021

Práctica 19: Listas doblemente ligadas.

Introducción:

Las listas ligadas dobles nos permiten tener una lista con elementos a los cuales se pueden acceder de principio de la lista a fin y de fin a principio, dependiendo del uso que se le quiera dar. Estas listas nos permiten hacer procesos mas sencillos, pero como su estructura necesita de nodos con ligas a la izquierda y derecha, estas listas requieren de mucha mas memoria que una lista normal de una sola liga a la derecha.

Desarrollo:

Esta practica consiste en agregarle funcionalidad aparte de la que ya tenia a la clase ListaEncadenadaDoble, esta clase se encuentra dentro del paquete estructuraslineales, a esta clase se le implementaron todos los métodos de la interface ListaDatos además de unos cuantos métodos más.

Como esta lista tiene el mismo funcionamiento que la lista encadenada simple, los métodos serán muy parecidos en cuanto a funcionalidad. Estos solo cambiaran en cuanto a el tipo de nodo que se utiliza, ya que la lista doble utiliza nodos dobles, además también se diferenciaran en que los métodos tendrán a ser mas fáciles, aunque en algunos casos estos métodos aumentaran en cuanto a líneas de código por el tipo de nodo que se utiliza y como estos están ligados a otros y otros nodos los ligan a ellos.

Como se implementa la interface ListaDatos, todos los métodos implementados se les deberá agregar funcionalidad, algunos métodos ya tenían funcionalidad por lo que ya no se pusieron en esta lista, los métodos a los que se les agrego funcionalidad son:

- Imprime la lista en forma inversa.
  ```
  public void imprimirOrdenInverso();
  ```
- Busca en el arreglo el elemento requerido.
  ```
  public Object buscar(Object elemento);
  ```
- Elimina el dato solicitado del arreglo.
  ```
  public Object eliminar(Object elemento);
  ```
- Compara la lista actual con una segunda lista.
  ```
  public boolean esIgual(Object listaDatos2);
  ```
- Modifica un elemento que se quiere.
  ```
  public boolean cambiar(Object elementoViejo,Object elementoNuevo, int
  ```

numVeces);

- Busca valores igual al parámetro.
  public ArregloDatos buscarValores(Object elemento);

- Vacía la lista.
  public void vaciar();

- Agrega los datos de la lista pasada como parámetro a la lista actual.
  public boolean agregarLista(Object listaDatos2);

- Invierte el orden de los elementos de la lista.
  public void invertir();

- Cuenta cuantos datos igual al elemento se encontraron.
  public int contar(Object elemento);

- Elimina los datos que tienen en común la lista actual y lista2.
  public boolean eliminarLista(Object listaDatos2);

- Agrega un elemento las veces indicadas.
  public void rellenar(Object elemento, int cantidad);

- Clona la lista actual.
  public Object clonar();

- Regresa una lista con los elementos indicados.
  public Object subLista(int indiceInicial, int indiceFinal);

- Rellena dependiendo si es un número, letra o un objeto.
  public void rellenar(Object elemento);

- Indica si la lista pasada como parámetro es sablista de la lista actual.
  public boolean esSublista(Object listaDatos2);

- Cambia los datos en común de la lista actual y la listaDatos2 y lo cambia por el elemento que está en listaDatos2Nuevos.
  public boolean cambiarLista(ArregloDatos listaDatos2,ArregloDatos listaDatos2Nuevos);

- Deja en la lista actual solo los elementos que se encuentran en listaDatos2.
  public boolean retenerLista(ArregloDatos listaDatos2);

- Inserta en la posición indicada el elemento.
  public boolean insertar(int indice, Object elemento);

- Agrega los datos de la lista a la lista actual quitando los anteriores.
  `public boolean copiarLista(ArregloDatos listaDatos2);`

Aparte de los métodos implementados por la interface, también se agregaron los siguientes métodos que se pedían:

- Inicia el iterador en el frente.
  `public void inicializarIterador()`

- Verifica si el iterador es null.
  `public boolean hayElementos()`

- Obtiene el elemento del iterador.
  `public Object obtenerElemento()`

- Inicia el iteradorOI en el fin.
  `public void inicializarIteradorOI()`

- Verifica si el iteradorOI es null.
  `public boolean hayElementosOI()`

- Obtiene el elemento del iteradorOI.
  `public Object obtenerElementoOI()`

- Atrasa los iteradores una vez si es que estos están iniciados.
  `public void atrasarIterador()`

- Inserta al inicio de la lista un elemento proporcionado.
  `public int agregarInicio(Object elemento)`

- Elimina el elemento del inicio si es que hay elementos en la lista.
  `public Object eliminarInicio()`

- Busca un elemento comenzando a buscar por el final.
  `public Object buscarOI(Object elemento)`

- Ingresa los elementos de la lista actual en una lista que separa los elementos por números, cadenas y otros.
  `public ListaEncadenadaDoble separarElementos()`

Como la lista doblemente encadenada utiliza nodos dobles los cuales tienen una liga izquierda y una derecha, todos estos métodos se tienen que cuidar bien que los nodos no queden apuntando a un nodo que se eliminó.

Nota: Toda la documentación del proyecto esta agregada en la carpeta “doc” dentro de la carpeta del proyecto (“edylab_2021_19/doc”).

Capturas del programa funcionando:

La clase de prueba es **“PruebaListaEncadenadaDoble”.** Esta clase se encuentra dentro del paquete “pruebas”. En esta clase se probo el funcionamiento de todos los metodos implementados y los que fueron agregados.

PruebaListaEncadenada:

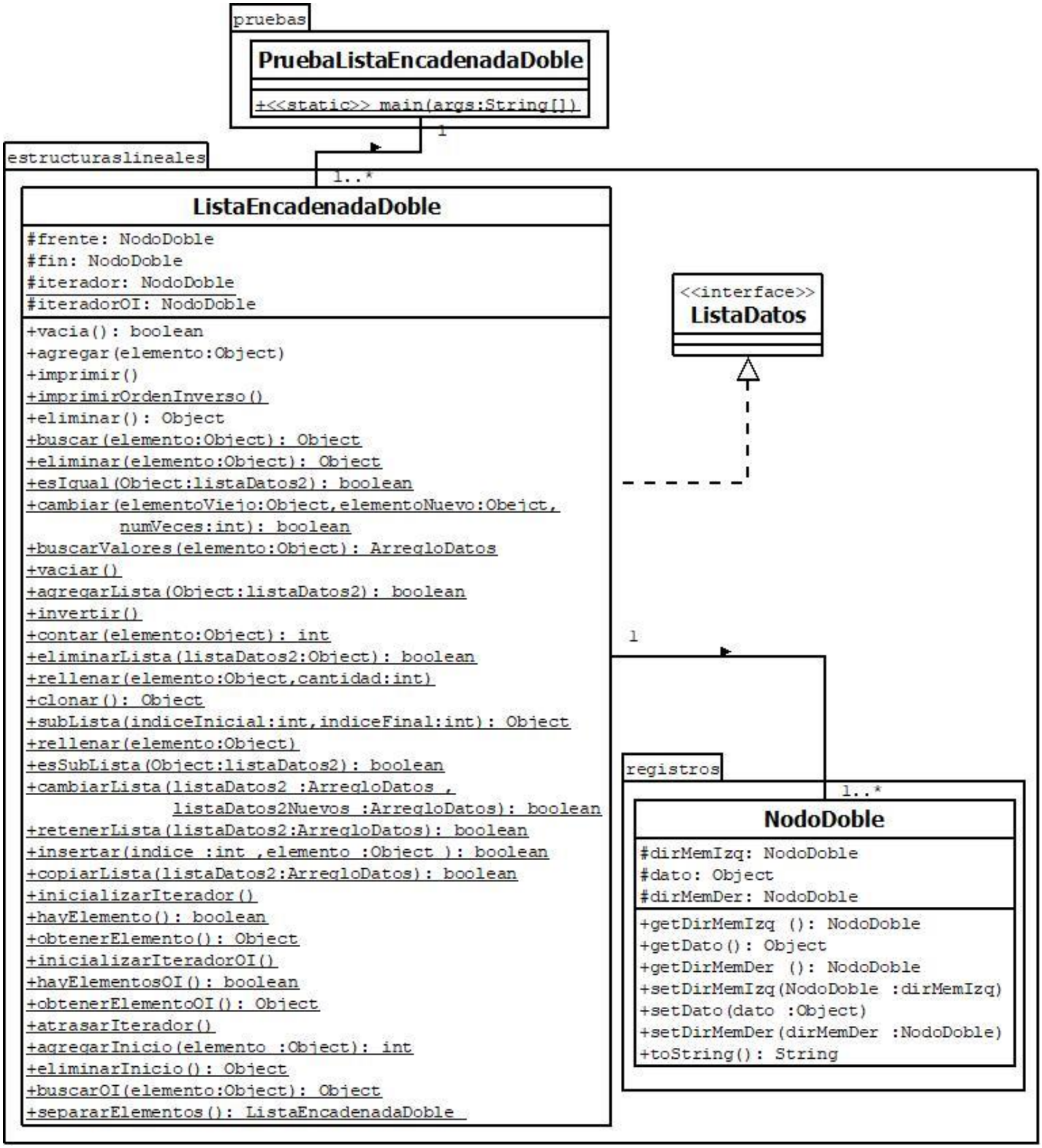
```
null <- A <--> B <--> C <--> D -> null
Imprimir en orden inverso:
null <- D <--> C <--> B <--> A -> null
Busca un elemento: C
Agregar los datos de una lista:
null <- A <--> B <--> C <--> D <--> E <--> F <--> G <--> H <--> X <--> A -> null
Copia de la lista:
null <- A <--> B <--> C <--> D <--> E <--> F <--> G <--> H <--> X <--> A -> null
Vaciar la lista:
null
Rellenar lista:
null <- X <--> X <--> X <--> X <--> X -> null
Contar coincidencias: 5
invertir:
null <- A <--> X <--> H <--> G <--> F <--> E <--> D <--> C <--> B <--> A -> null
Cambiar: true
null <- K <--> K <--> K <--> X <--> X -> nullComparar listas: true

Eliminar elemento: null
null <- A <--> X <--> H <--> G <--> F <--> E <--> D <--> C <--> B <--> A -> null
Buscar valores:
X
X
null <- J <--> E <--> A <--> S <--> F <--> G -> null
Eliminar lista: true
null <- J <--> S <--> G -> null
```

```
Sub lista:
null <- S <--> G <--> E -> null
Es sublista?: false
Cambiar lista: true
null <- J <--> S <--> G <--> Z <--> Y <--> X -> null
Retener lista: true
null <- X <--> Y <--> Z -> null
Insertar: true
null <- X <--> K <--> Z -> null
Copiar lista: true
null <- F <--> A <--> E -> null
null <- X <--> 1 <--> 987656 <--> 54 <--> Y <--> Palabra <--> 123 <--> Hola -> null
Eliminar primer elemento: X
null <- 1 <--> 987656 <--> 54 <--> Y <--> Palabra <--> 123 <--> Hola -> null
Buscar comenzando del final: 54
Separar elementos de la lista:
null <- 1 <--> 54 <--> 123 -> null
null <- Y <--> Hola -> null
null <- 987656 <--> Palabra -> null
```

Código agregado:

Los atributos y métodos subrayados son nuevos, además de la clase PruebaListaEncadenadaDoble.


pruebas
PruebaListaEncadenadaDoble
+<<static>> main(args:String[])
1
estructuraslineales
1..*
ListaEncadenadaDoble
#frente: NodoDoble
#fin: NodoDoble
#iterador: NodoDoble
#iteradorOI: NodoDoble
+vacia(): boolean
+agregar(elemento:Object)
+imprimir()
+imprimirOrdenInverso()
+eliminar(): Object
+buscar(elemento:Object): Object
+eliminar(elemento:Object): Object
+esIgual(Object:listaDatos2): boolean
+cambiar(elementoViejo:Object,elementoNuevo:Obejct,
numVeces:int): boolean
+buscarValores(elemento:Object): ArregloDatos
+vaciar()
+agregarLista(Object:listaDatos2): boolean
+invertir()
+contar(elemento:Object): int
+eliminarLista(listaDatos2:Object): boolean
+rellenar(elemento:Object,cantidad:int)
+clonar(): Object
+subLista(indiceInicial:int,indiceFinal:int): Object
+rellenar(elemento:Object)
+esSubLista(Object:listaDatos2): boolean
+cambiarLista(listaDatos2 :ArregloDatos ,
listaDatos2Nuevos :ArregloDatos): boolean
+retenerLista(listaDatos2:ArregloDatos): boolean
+insertar(indice :int ,elemento :Object): boolean
+copiarLista(listaDatos2:ArregloDatos): boolean
+inicializarIterador()
+hayElemento(): boolean
+obtenerElemento(): Object
+inicializarIteradorOI()
+hayElementosOI(): boolean
+obtenerElementoOI(): Object
+atrasarIterador()
+agregarInicio(elemento :Object): int
+eliminarInicio(): Object
+buscarOI(elemento:Object): Object
+separarElementos(): ListaEncadenadaDoble
<<interface>>
ListaDatos
1
registros
1..*
NodoDoble
#dirMemIzq: NodoDoble
#dato: Object
#dirMemDer: NodoDoble
+getDirMemIzq (): NodoDoble
+getDato(): Object
+getDirMemDer (): NodoDoble
+setDirMemIzq(NodoDoble :dirMemIzq)
+setDato(dato :Object)
+setDirMemDer(dirMemDer :NodoDoble)
+toString(): String

Pre-evaluación:

Pre-Evaluación para prácticas de Laboratorio de Estructuras de Datos	PRE-EVALUACIÓN DEL ALUMNO
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.	Sí
INCLUYE LA DOCUMENTACIÓN GENERADA CON JAVADOC.	Sí
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).	100%
Observaciones:	

Conclusión:

Usar una lista doblemente ligada nos ayuda a la hora de acceder a los datos y así manipularlos de mejor forma, pero también nos perjudican a la hora de usar muchísimos datos ya que esto utiliza más memoria además de que algunos procesos pueden ser más complicados y que utilizan muchos más procesos comparados con los que utiliza una lista simplemente ligada.