Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software.

Nombre de la Práctica Listas simplemente ligadas.

Numero de Práctica 16

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: 30/09/2021

Práctica 16: Listas simplemente ligadas.

Introducción:

Las listas que utilizan memoria dinámica son de gran utilidad a la hora de almacenar una gran cantidad de elementos en la que la cantidad de elementos es indefinida, ya que gracias a la memoria dinámica podemos almacenar tantos objetos como nos permita la memoria RAM. En cambio, con los arreglos, las listas no contienen un índice con el cual acceder a los datos, si no que se tiene que ir recorriendo la lista para poder obtener el dato querido.

Desarrollo:

Como la clase ListaEncadenada implementa la interface ListaDatos, todos los métodos que contiene esta interface los tiene que tener la clase ListaEncandenada, por lo que en esta práctica se le estará agregando funcionalidad a estos métodos implementados, además de agregar algunos métodos más.

Para esto se hizo uso de la clase ListaEncadenada que se encuentra dentro del paquete (src/estructuraslineales/), en ella se le agrego funcionalidad a los siguientes métodos:

ListaEncadenada:

- 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);
  ```

- Este método no hace nada por su estructura.
  ```
  public void rellenar(Object elemento);
  ```

- Indica si la lista pasada como parámetro es sublista 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);
  ```

Y también se agregaron los siguientes métodos que no son implementados de la interface ListaDatos:

- Agrega todos los elementos de la lista a un arreglo.
  ```
  public ArregloDatos aArreglo();
  ```

- Agrega todos los elementos de la lista que no se encuentren en el arreglo.
  ```
  public ArregloDatos aArreglo(ArregloDatos arregloADescartar);
  ```

- Pasa los datos de la lista a una matriz2D, si faltan datos para rellenar la matriz se agregan null.

```
public Tabla2D aTabla2D(int filas, int columnas)

• Agrega la matriz columna por columna o fila por fila según la opción del
  enumerado.
  public boolean agregarTabla2D(Tabla2D tabla, TipoTabla enumTipoTabla) ;

• Cambia un objeto dada una posición en la lista.
  public boolean cambiar(int indice, Object elemento);

• Obtiene el elemento del índice indicado.
  public Object obtener(int indice);

• Agrega o quita elementos de la lista.
  public boolean redimensionar(int maximo);

• Elimina el elemento que se encuentra en la posición dada.
  public Object eliminar(int indice);
```

Casi todos los métodos se pudieron hacer. No fueron todos ya que uno de estos no se podía implementar debido a su estructura, este método es “rellenar(Object elemento)”, este método se podía implementar en los arreglos ya que estos si tienen un limite de elementos que pueden admitir, en cambio la lista no lo tiene y solo pararía cuando la memoria se desborde, por lo cual se decidió no implementar funcionalidad en este método, pero se tuvo que dejar por la implementación que conlleva.

Además de esto, un método hace uso de un enumerado aun no creado, por lo que se tuvo que hacer, este enumerado se creó en el paquete (src/estructrasnolineales/), ya que pertenece a las tablas, este enumerado se llama “TipoTabla”, el cual tiene las opciones de “COLUMNA” y “FILA”. El método que utiliza este enumerado es “agregarTabla2D(Tabla2D tabla, TipoTabla enumTipoTabla)”, el cual utiliza el enumerado para determinar como se iran agregando los datos, si será fila por fila o columna por columna.

Nota: Toda la documentación del proyecto esta agregada en la carpeta “doc” dentro de la carpeta del proyecto (“edylab_2021_16/doc”).

Capturas del programa funcionando:

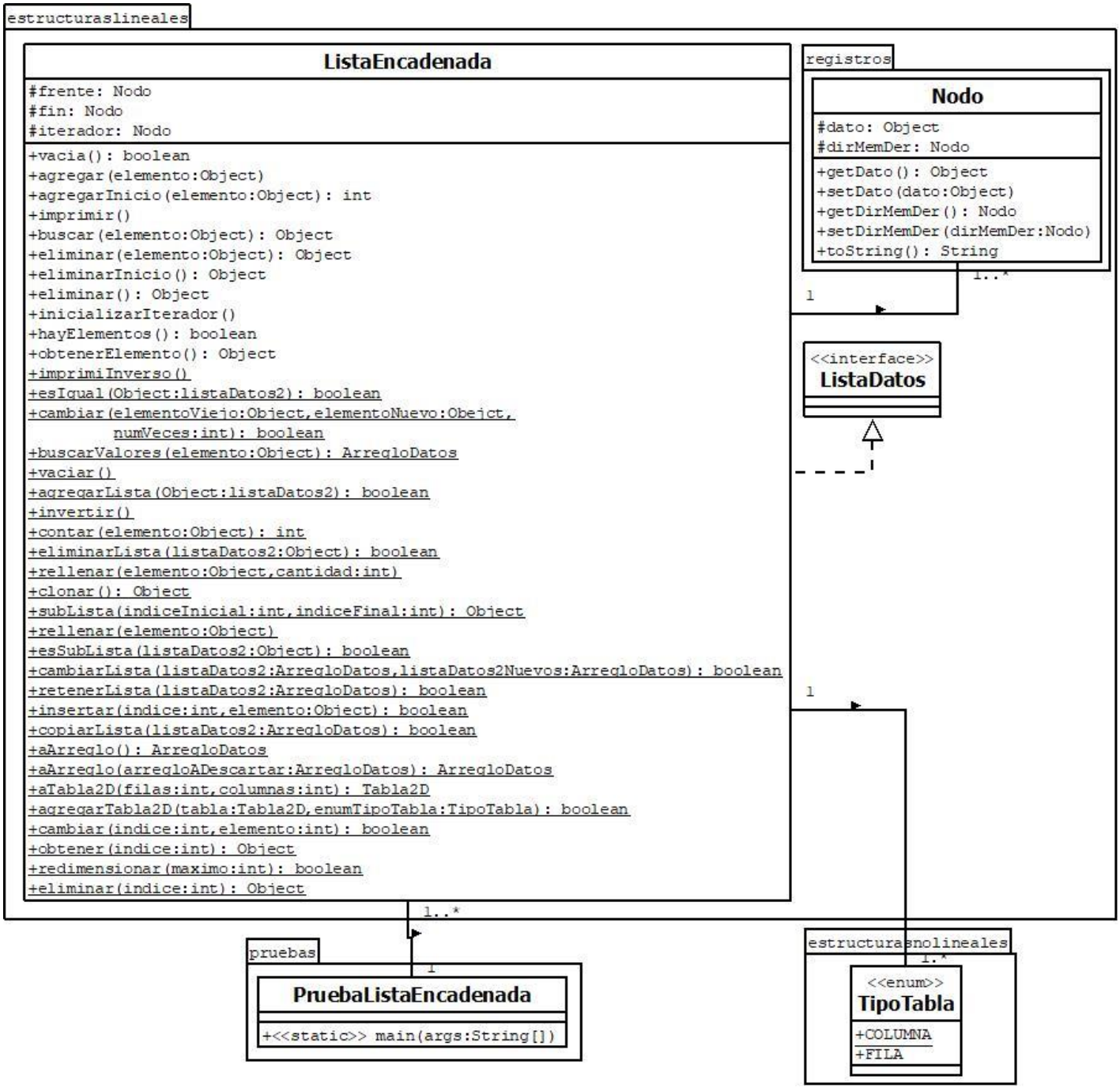
```
A -> B -> C -> D -> null
Imprimir en orden inverso:
null <- D <- C <- B <- A
Busca un elemento: C
Lista a arreglo:
A
B
C
D
Lista a arreglo sin repetidos por el arreglo
E
F
G
Lista a matriz2D:
A B C
D E F
G null null
Agregar los datos de una lista:
A -> B -> C -> D -> E -> F -> G -> A -> B -> C -> D -> null
Copia de la lista:
A -> B -> C -> D -> E -> F -> G -> A -> B -> C -> D -> null
Agregar matriz a lista: true
A -> B -> C -> D -> E -> F -> G -> A -> B -> C -> D -> A -> B -> C -> D -> E -> F -> G -> null -> null -> null
Vaciar la lista:
null
```

```
Rellenar lista:
X -> X -> X -> X -> X -> null
Contar coincidencias: 5
invertir:
null -> null -> G -> F -> E -> D -> C -> B -> A -> D -> C -> B -> A -> G -> F -> E -> D -> C -> B -> A -> null
Cambiar: true
K -> K -> K -> X -> X -> null
Cambiar por indice: true
K -> K -> K -> Ñ -> X -> null
Obtener por indice: Ñ
Comparar listas: true
Redimensionar: true
null -> null -> G -> F -> null
Eliminar por indice: G
null -> null -> F -> null
Buscar valores:
K
K
K
J -> E -> A -> S -> F -> G -> null
Eliminar lista: true
J -> S -> G -> null
Sub lista:
S -> G -> E -> null
Es sublista?: true
```

```
Cambiar lista: true
J -> S -> G -> Z -> Y -> X -> null
Retener lista: true
X -> Y -> Z -> null
Insertar: true
X -> K -> Z -> null
Copiar lista: true
F -> A -> E -> null
```

Código agregado:

Todos los métodos subrayados son nuevos o fue a los que se agregó funcionalidad, solo se creó el enumerado TipoTabla, todas las demás clases ya estaban creadas



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.	No
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:

Las listas encadenadas son mucho más fáciles de manejar fuera de estas, ya que no pide ningún requisito para poder crearlas y agregarle o eliminarle datos, pero ya dentro de estas, los métodos suelen ser un poco más complicados por la forma en la que se maneja la lista, ya que se tienen que recorrer la lista para cualquier acción que se le quiera realizar a la lista o algún dato de la lista en específico (excepto agregar). Como los datos que se almacenan no tienen un índice con el cual se pueda acceder a ellos, es más complicado que ejecute métodos que requieran índices ya que para esto se tiene que recorrer la lista para saber si este sobrepasa los límites de la lista o no.