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 17

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: 01/10/2021

Práctica 17: Listas simplemente ligadas.

Introducción:

Las listas ordenadas nos permiten almacenar una gran cantidad de elementos los cuales estarán en el orden que se escoja (Ascendente o Descendente). Esto tiene una gran utilidad a la hora de darle un orden a los datos que se vallan a agregar a la lista. Para cualquier persona le resulta mucho más fácil identificar un elemento que esta en una lista ordenada ya que este no puede ser mayor o menor que otros elementos, por lo que si se buscara un elemento en una lista desordenada seria mucho mas complicado para cualquier persona.

Desarrollo:

Para el desarrollo de esta práctica se estuvo haciendo uso de la herencia ya que se heredo de la clase ListaEncadenada para solo tener que sobrescribir los métodos que pueden incumplir el orden de los datos. El método mas importante para realizar la clase ListaEncadenadaOrden es el de agregar, ya que sin este ningún otro método funcionaria, ya que muchos métodos utilizan este para poder agregar datos a la lista sin tener que repetir código.

Los métodos de eliminar no se tuvieron que sobrescribir ya que no necesitan saber el orden, lo único que necesitan es saber si existen los datos que se van a eliminar, para una lista ordenada no necesitan más funcionalidad.

ListaEncadenadaOrden: Esta clase hereda de la clase ListaEncadenada y se encuentra en el mismo paquete que esta (src/estructuraslineales/), y solo se sobrescribieron los siguientes métodos:

- Agrega de forma ordenada el dato, dependiendo del tipo de orden que se escoja.
  `public int agregar(Object elemento)`
- No agrega nada ya que perdería el orden.
  `public int agregarInicio(Object elemento)`
- Busca un dato en la lista.
  `public Object buscar(Object elemento)`
- Elimina el elemento de la posición indicada y agrega el elemento de forma ordenada.
  `public boolean cambiar(int indice, Object elemento)`

- Elimina el elemento viejo y agrega el nuevo de forma ordenada.
  `public boolean cambiar(Object elementoViejo, Object elementoNuevo, int numVeces)`
- Invierte la lista así como el orden que tiene.
  `public void invertir()`
- Agrega el elemento una sola vez sin importar la cantidad.
  `public void rellenar(Object elemento, int cantidad)`
- Agrega el elemento ordenadamente una sola vez.
  `public void rellenar(Object elemento)`
- Agrega un dato en la lista, sin importar la posición en la que se quiera.
  `public boolean insertar(int indice, Object elemento)`

Como en las listas ordenadas no se permiten elementos duplicados y además de eso las listas no tienen un límite definido de elementos, los métodos de rellenar lo único que hacen es agregar el elemento que se pasa como argumento una sola vez, con esto se logra tener el orden de la lista tal y como debe de ser.

Para lograr que el método de agregar funcione para los dos tipos de orden que se pueden tener, el método evalúa todas las posibilidades en las que se puede agregar un elemento en una posición o en otra, como saber si se agrega en el frente o después del frente, o si se agrega al final o antes de este, eso ya dependerá de lo que arroje la comparación de los datos. Esto nos da como resultado una gran variedad de if-else donde si no es una cosa es otra, pero siempre tiene que agregar el elemento si es que aun no esta en la lista, si el elemento ya esta en la lista no lo vuelve a agregar, con esto mantenemos el orden y que no haya duplicados.

Nota: Toda la documentación del proyecto esta agregada en la carpeta “doc” dentro de la carpeta del proyecto (“edylab_2021_17/doc”).

Capturas del programa funcionando:

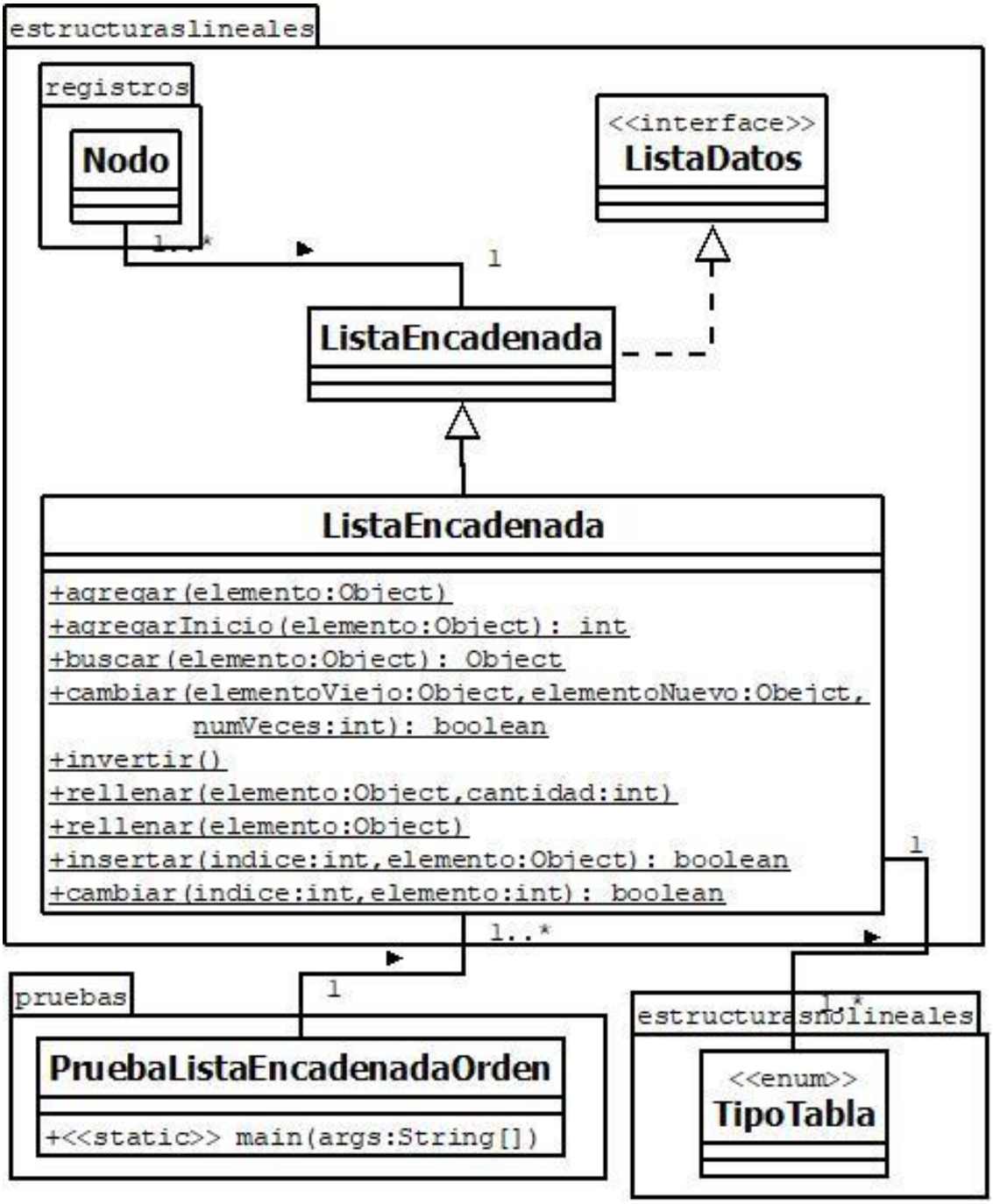
Para la prueba de que la lista funciona tal y como deberia, esta hace dos listas, una en un orden ascendente y la otra en orden descendente y se prueban solo los metodos sobrescritos y alguno que otro metodo heredado.

```
1 -> 12345678 -> 2 -> 15 -> A -> ADIOS -> H -> I -> K -> S -> null
Imprimri inverso
null <- S <- K <- I <- H <- ADIOS <- A <- 15 <- 2 <- 12345678 <- 1
Agregar una tabla2D: true
0 -> 1 -> 12345678 -> 2 -> 15 -> 24 -> A -> ADIOS -> H -> I -> K -> S -> Ñ -> null
Agregar arreglo: true
0 -> 1 -> 12345678 -> 2 -> 15 -> 24 -> A -> ADIOS -> E -> EZZ -> H -> I -> K -> Q -> S -> Ñ -> null
Agregar un elemento en una posicion indicada: true
0 -> 1 -> 12345678 -> 2 -> 15 -> A -> ADIOS -> B -> E -> EZZ -> H -> I -> K -> Q -> S -> Ñ -> null
Es igual: true
Cambiar viejo por nuevo: true
0 -> 1 -> 12345678 -> 2 -> 15 -> A -> B -> E -> EZZ -> H -> HOLA -> I -> K -> Q -> S -> Ñ -> null
Invertir lista:
Ñ -> S -> Q -> K -> I -> HOLA -> H -> EZZ -> E -> B -> A -> 15 -> 2 -> 12345678 -> 1 -> 0 -> null
Rellenar:
# -> Ñ -> S -> Q -> K -> I -> HOLA -> H -> EZZ -> E -> B -> A -> 15 -> 2 -> 12345678 -> 1 -> 0 -> null
Cambiar lista: true
# -> 1 -> Ñ -> S -> Q -> K -> I -> HOLA -> H -> EZZ -> E -> B -> A -> 15 -> 2 -> 12345678 -> 0 -> null
```

```
LISTA CON ORDEN DESCENDENTE
S -> K -> I -> H -> ADIOS -> A -> 15 -> 2 -> 12345678 -> 1 -> null
Imprimri inverso
null <- 1 <- 12345678 <- 2 <- 15 <- A <- ADIOS <- H <- I <- K <- S
Agregar una tabla2D: true
Ñ -> S -> K -> I -> H -> ADIOS -> A -> 24 -> 15 -> 2 -> 12345678 -> 1 -> 0 -> null
Agregar arreglo: true
Ñ -> S -> Q -> K -> I -> H -> EZZ -> E -> ADIOS -> A -> 24 -> 15 -> 2 -> 12345678 -> 1 -> 0 -> null
Agregar un elemento en una posicion indicada: true
Ñ -> S -> Q -> K -> I -> EZZ -> E -> B -> ADIOS -> A -> 24 -> 15 -> 2 -> 12345678 -> 1 -> 0 -> null
Es igual: true
Cambiar viejo por nuevo: true
Ñ -> S -> Q -> K -> I -> HOLA -> EZZ -> E -> B -> A -> 24 -> 15 -> 2 -> 12345678 -> 1 -> 0 -> null
Invertir lista:
0 -> 1 -> 12345678 -> 2 -> 15 -> 24 -> A -> B -> E -> EZZ -> HOLA -> I -> K -> Q -> S -> Ñ -> null
Rellenar:
0 -> 1 -> 12345678 -> 2 -> 15 -> 24 -> A -> B -> E -> EZZ -> HOLA -> I -> K -> Q -> S -> Ñ -> # -> null
Cambiar lista: true
1 -> 0 -> 12345678 -> 2 -> 15 -> 24 -> A -> B -> E -> EZZ -> HOLA -> I -> K -> Q -> S -> Ñ -> # -> null
```

Código agregado:

La clase ListaEncadenadaOrden es nueva, al igual que los métodos dentro de esta clase. También la clase PruebaListaEncadenadaOrden es nueva y la cual contiene el main para ejecutar el programa.



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 ordenadas son muy parecidas a los arreglos ordenados porque estos tienen un orden definido, en lo único que se diferencian es en que el arreglo tiene una cantidad de elementos definido y no pude cambiar, en cambio la lista al utilizar memoria dinámica puede ir agregando elementos hasta que haya un desbordamiento de memoria.

Los arreglos y las listas ordenadas hacen lo mismo, que es agregar elementos en una posición definida al ver que elemento es más grande o cual es más pequeño que el que se va agregar dependiendo de que tipo de orden se escoja.