Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software.

Nombre de la Práctica Listas ligadas.

Numero de Práctica 20

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: 07/10/2021

Práctica 20: Listas 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:

Para el desarrollo de esta practica se tuvo que hacer una nueva lista. Esta lista contiene nodos nuevos por lo que no se pude usar la herencia para heredar de ListaEncadenada, tampoco se podrá hacer uso de la interface ListaDatos ya que los métodos que contiene son diferentes a los que contiene esta interface. Para esta lista se hizo una nueva interface llamada IListaHash.

La nueva lista se llamará ListaEncadenadaHash, la cual almacenará nodos de tipo NodoHash. Estos nodos son muy particulares ya que almacenan una clave, el valor (objeto a almacenar) y su liga derecha. La clave será para identificar algún elemento en la lista y no podrá haber duplicados, mientras que en el campo valor si podrá haber duplicados. La clase ListaEncadenadaHash se agrego en el paquete estructuraslineales, y la clase NodoHash se agrego dentro del paquete registros que esta dentro del paquete estructuraslineales.

El campo valor de la clase NodoHash será de tipo Object para poder almacenar cualquier tipo de dato en la lista. El campo clave será de tipo Object por lo que podrá almacenar cualquier tipo de dato que se le proporcione y como este es un identificador para el elemento almacenado siempre se estarán haciendo comparaciones, al menos en los métodos más esenciales como el de insertar, esto ocasionará que se utilice en muchas ocasiones la clase Herramientas con su método compararObjetos(Object objeto1, Object objeto2).

Los métodos de esta lista harán cosas muy similares a los métodos que se han estado haciendo en los demás tipos de listas que se tienen, por lo que el funcionamiento será el mismo, solo cambiando algunas cosas por el tipo de nodo que se utiliza.

Este tipo de nodo nos permite tener métodos de la lista más cortos o más largos en comparación con otras listas, esto dependiendo de que es lo que se valla hacer.

Los métodos que se agregaron a esta lista son:

- Evalúa si la lista está vacía.
  ```
  public boolean vacia();
  ```

- Inserta un elemento al final de la lista, cambia el valor si existe una clave repetida.
  ```
  public boolean insertar(Object clave, Object valor);
  ```

- Elimina un dato de la lista por la clave que tenga.
  ```
  public Object eliminar(Object clave);
  ```

- Elimina un dato de la lista por el valor que tenga.
  ```
  public Object eliminarValor(Object valor);
  ```

- Busca un elemento de la lista por la clave.
  ```
  public Object buscar(Object clave);
  ```

- Busca un elemento en la lista por el valor.
  ```
  public Object buscarValor(Object valor);
  ```

- Sustituye un valor de la lista buscando por la clave que tiene.
  ```
  public boolean substituir(Object clave, Object valorNuevo);
  ```

- Sustituye un valor de la lista buscando por el valor que tiene.
  ```
  public boolean substituirValor(Object valor, Object valorNuevo);
  ```

- Imprime la lista.
  ```
  public void imprimir();
  ```

- Imprime las claves que tiene de la lista.
  ```
  public void imprimirClaves();
  ```

- Imprime los valores que contiene la lista.
  ```
  public void imprimirValores();
  ```

- Agrega las claves a un arreglo y los valores a otro arreglo y estos arreglos los agrega una lista ligada.
  ```
  public ListaEncadenada aArreglos();
  ```

- Agrega las claves de la lista a una lista ligada y los valores de la lista a otra lista ligada, y estas listas las agrega a otra lista.
  ```
  public ListaEncadenada aListas();
  ```

- Agrega la lista a una matriz, donde en una columna agrega las claves y en otra columna agrega los valores.
  `public Tabla2D aTabla();`

- Vacía la lista.
  `public void vaciar();`

- Obtiene un elemento de la lista por la clave.
  `public Object obtener(Object clave);`

- Agrega todos los elementos de la lista2 a la lista actual.
  `public boolean agregarTodos(ListaEncadenadaHash lista2);`

- Obtiene la longitud de la lista.
  `public int cantidadElementos();`

- Agrega a la lista actual las claves y los valores que contienen los arreglos paralelos.
  `public boolean agregarArreglos(ArregloDatos arregloClaves, ArregloDatos arregloValores);`

- Agrega a la lista actual las claves y los valores que contienen las listas paralelas.
  `public boolean agregarListas(ListaEncadenada listaClaves, ListaEncadenada listaValores);`

- Agrega los datos de la matriz a la lista actual, la primera columna son las claves y la segunda los valores.
  `public boolean agregarTabla(Tabla2D tabla);`

Los métodos mas importantes que son los primeros 9 métodos, y son los métodos que mas se utilizan, tanto los utilizan otros métodos de la misma lista o son los mas importantes a la hora de hacer alguna actividad con una lista. Estos métodos utilizan en la gran mayoría la clave de los nodos para realizar alguna acción sobre la lista.

Nota: Toda la documentación del proyecto esta agregada en la carpeta “doc” dentro de la carpeta del proyecto (“edylab_2021_20/doc”).

Capturas del programa funcionando:

La clase **PruebaListaEncadenadaHash** se encuentra dentro del paquete pruebas.

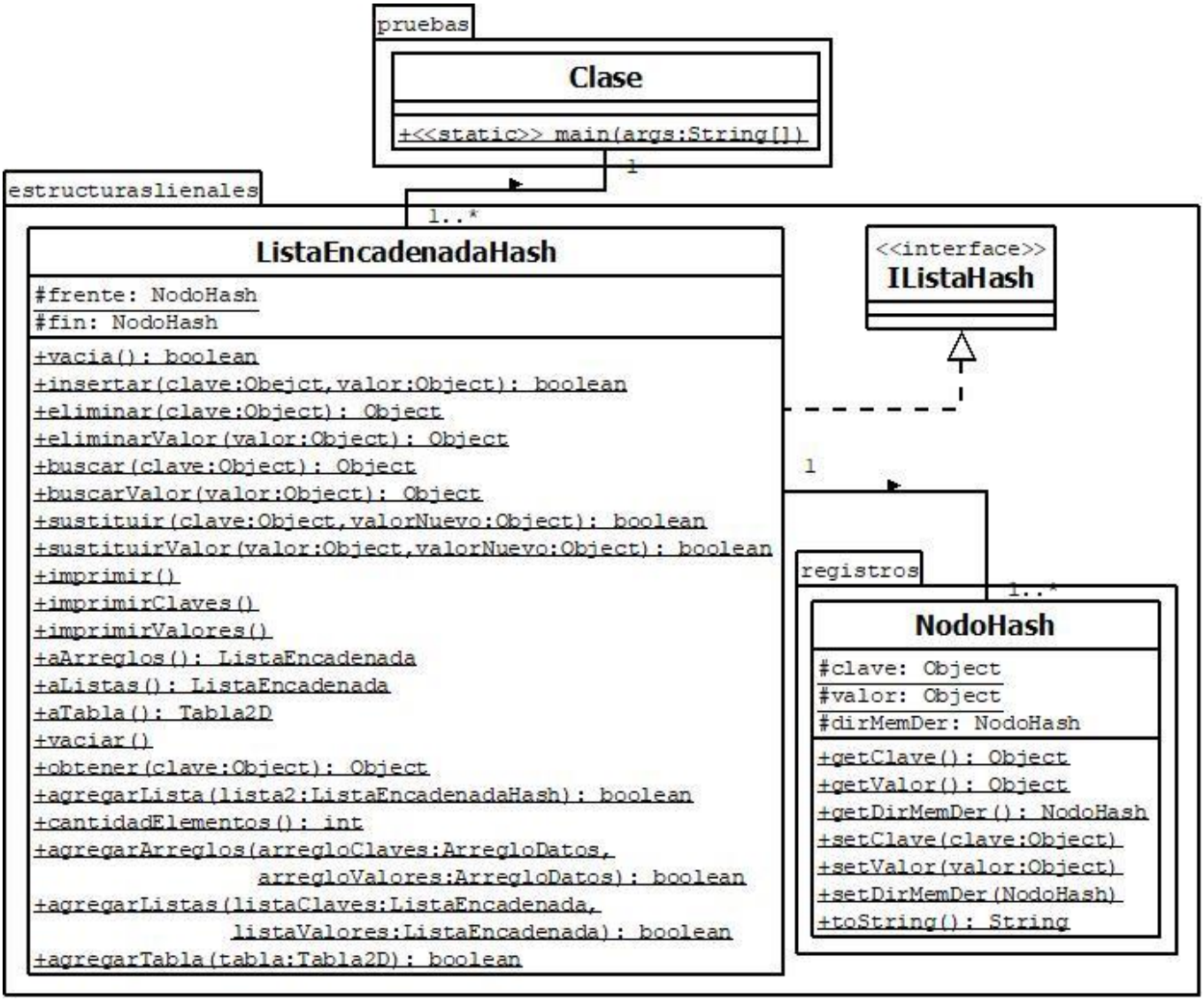
```
Lista 1:
1-A -> 2-B -> 3-C -> 4-D -> 5-F -> 6-J -> 7-D -> null
Eliminar por clave: A
2-B -> 3-C -> 4-D -> 5-F -> 6-J -> 7-D -> null
Eliminar por valor: D
2-B -> 3-C -> 4-D -> 5-F -> 6-J -> null
Buscar por clave: F
Buscar por valor: B
Substituir por clave: true
2-B -> 3-Z -> 4-D -> 5-F -> 6-J -> null
Substituir por valor: : true
2-B -> 3-Z -> 4-D -> 5-F -> 6-H -> null
Imprimir claves:
2 -> 3 -> 4 -> 5 -> 6 -> null
Imprimir valores:
B -> Z -> D -> F -> H -> null
A arreglos:
Claves:
2
3
4
5
6
```

```
Valores:
B
Z
D
F
H

A listas ligadas:
Claves:
2 -> 3 -> 4 -> 5 -> 6 -> null
Valores:
B -> Z -> D -> F -> H -> null
A matriz2D:
2 B
3 Z
4 D
5 F
6 H

Vaciar l2:
1-A -> 2-B -> 3-C -> 4-D -> 5-F -> 6-J -> 7-D -> null
null
Obtener por clave: D

Agregar todo lista:
2-B -> 3-Z -> 4-Ñ -> 5-F -> 6-H -> 0-P -> 8-S -> 10-M -> null
Longitud de la lista: 8
```

```
l2:
1-Z -> 2-X -> 3-G -> null
Agregar arreglos a l2:
1-Z -> 2-B -> 3-Z -> 4-D -> 5-F -> 6-H -> null
l2:
1-A -> 2-F -> 3-Ñ -> null
Agregar listas a l2:
1-A -> 2-B -> 3-Z -> 4-D -> 5-F -> 6-H -> null
l2:
1-N -> 2-V -> 3-T -> 4-D -> 5-F -> 6-H -> null
Agregar Matriz a l2:
1-N -> 2-B -> 3-Z -> 4-D -> 5-F -> 6-H -> null
```

Código agregado:

Todas las clases que se miran en el diagrama son nuevas, además de todos sus métodos. La interface IListaHash contiene definidos todos los métodos que se encuentran dentro de la clase ListaEncadenadaHash.



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 que contiene identificadores para cada elemento que contiene, es muy útil a la hora de identificar los valores de la lista.

Aunque estas listas requieren de un poco más de memoria que una lista encadenada simple, ya que almacenan un objeto para la clave y uno para el valor que se va a almacenar, para muchos de los procesos de manipulación de los datos se requieren menos acciones para obtener lo que queramos, como por ejemplo a la hora de buscar es más fácil hacerlo con un identificador del elemento.