LISTAS ENLAZADAS

(1) INTRODUCCIÓN:

Una lista enlazada o estructura ligada, es una estructura lineal que almacena una colección de elementos generalmente llamados nodos, en donde cada nodo puede almacenar datos y ligarlos a otros nodos. De esta manera los nodos pueden localizarse en cualquier parte de la memoria, utilizando la referencia que lo relaciona con otro nodo dentro de la estructura.

Las listas enlazadas son estructuras dinámicas que se utilizan para almacenar datos que están cambiando constantemente. A diferencia de los vectores, las estructuras dinámicas se expanden y se contraen haciéndolas más flexibles a la hora de añadir o eliminar información.

Las listas enlazadas permiten almacenar información en posiciones de memoria que no sean contiguas; para almacenar la información contienen elementos llamados nodos. Estos nodos poseen dos campos: uno para almacenar la información o valor del elemento y otro para el enlace que determina la posición del siguiente elemento o nodo de la lista.

```
public class Nodo {
      Object elemento;
      Nodo siguiente;
}
```

Las operaciones que se pueden hacer con una lista son:

- Inserción de un elemento
- Borrado de un elemento
- Recorrido de una lista
- Búsqueda de un elemento

(2) LISTAS ENLAZADAS SIMPLES:

Una lista enlazada simple es una colección de nodos que tienen una sola dirección y que en conjunto forman una estructura de datos lineal. Cada nodo es un objeto compuesto que guarda una referencia a un elemento (dato) y una referencia a otro nodo (dirección).

La referencia que guarda un nodo a otro nodo se puede considerar un enlace o un puntero hacia el segundo nodo y el salto que los relaciona recibe el nombre de salto de enlace o salto de puntero. El primer nodo de una lista recibe el nombre de cabeza, cabecera o primero y el último es llamado final, cola o último (es el único nodo con la referencia a otro objeto como nula).

Un nodo de una lista enlazada simple puede determinar quien se encuentra después de él pero no puede determinar quien se encuentra antes, ya que solo cuenta con la dirección del nodo siguiente pero no del anterior.

(a) Operaciones:

INSERCIÓN DE ELEMENTOS		
Al inicio de la lista	**En alguna posición específica**	**Al final de la lista**
Verificar si la lista está vacía (i) Si está vacía la lista, es decir, inicio == null - Se crea un nuevo nodo o se asigna un nuevo nodo al inicio - Final se hace que apunte también al inicio ya que es el único nodo (ii) Si ya existe por lo menos un nodo - Se crea un nodo que se hace que apunte al inicio nuevo.sig=inicio - Se hace que inicio ahora apunte al primer elemento que es nuevo inicio=nuevo	Verificar si la lista está vacía o posición es igual a cero (i) Si está vacía la lista, es decir, inicio == null - Se crea un nuevo nodo o se asigna un nuevo nodo al inicio - Final se hace que apunte también al inicio ya que es el único nodo (ii) Si ya existe por lo menos un nodo - Se recorre la lista contando las posiciones de los elementos dentro de la lista (pos++), además se debe de recorrer conociendo el nodo anterior y el siguiente con el fin de reorganizar la lista al insertar un nuevo elemento - Si se localiza la posición buscada y siguiente es diferente de null (`if(cont==pos && sig!=null)`), se crea un nuevo nodo y se hace que apunte al siguiente nodo y además que el nodo anterior apunte al nuevo nodo. nuevo.sig=sig y ant.sig=nuevo - Si no se localiza la posición (indica que se dio una posición mayor a la cantidad de elementos, entonces se agrega el elemento al final de la lista	Verificar si la lista está vacía (i) Si está vacía la lista, es decir, inicio == null - Se crea un nuevo nodo o se asigna un nuevo nodo al inicio - Final se hace que apunte también al inicio ya que es el único nodo (ii) Si ya existe por lo menos un nodo - Se crea o se asigna un nuevo nodo a fin.sig=nuevo - Final se hace que apunte al nuevo nodo ya que ahora será el final

ELIMINACIÓN DE ELEMENTOS		
Del inicio de la lista	**De alguna posición específica**	**Del final de la lista**
Verificar si la lista está vacía (i) Si hay un solo nodo - Se pone ini=fin=null (ii) Si hay más de un nodo - El inicio es ahora el siguiente ini=ini.sig	Verificar si la lista está vacía (i) Si hay un solo nodo - Se pone ini=fin=null (ii) Si hay más de un nodo - Se debe recorrer la lista llevando el anterior, siguiente y el contador de las posiciones mientras que siguiente sea diferente de null. - Si cont==0 entonces se elimina del inicio - En otro caso si cont==pos y además sig!=null se pone a anterior.sig la dirección de siguiente.sig para eliminar a siguiente. - También hay que verificar si es el último elemento para ponerlo como final si anterior.sig==null entonces se debe poner como final (fin=ant) - Si no se localiza la posición se elimina del final de la lista.	Verificar si la lista está vacía (i) Si hay un solo nodo - Se pone ini=fin=null (ii) Si hay más de un nodo - Se debe recorrer la lista llevando el anterior y el siguiente mientras que el anterior.sig sea diferente del final o del siguiente (debe de construir primero el nodo anterior para poder comparar con siguiente) - Al llegar al final entonces el anterior se debe poner como final y a final.sig como null

(3) ARRAYLIST:

La clase ArrayList, permite crear una estructura de datos con bloques de memoria continuos similar a un arreglo unidimensional. El objeto creado a partir de la clase ArrayList es por lo tanto un arreglo flexible; es decir, un arreglo que puede cambiar su tamaño de forma dinámica. Esta estructura puede ser ideal para el manejo de pilas, colas y listas, y se encuentra definida en la librería `util`.

(a) Constructores de la clase ArrayList:

ArrayList lista=new ArrayList()	Crea un objeto llamado lista de una capacidad inicial de 10 elementos
ArrayList lista=new ArrayList(int cap)	Crea una lista de capacidad inicial indicada por un valor de tipo entero (se lanza la excepción *`IllegalArgumentException`* si la capacidad es inferior a 1)
ArrayList lista=new ArrayList(Collection col)	Crea una lista con la colección de elementos que se pasan como argumento apoyándose de la clase `Collection`

(b) Métodos de la clase ArrayList:

add(Object elem)	Inserta un elemento al final de la lista retornando un valor booleano si se pudo insertar o no
add(int pos, Object elem)	Inserta un elemento en la posición indicada desplazando los demás elementos una posición hacia el final de la lista, lanza la excepción *`IndexOutOfBoundsException`* si la posición está fuera del tamaño de la lista
clean()	Elimina todos los elementos de la lista
clone()	Regresa una copia de la lista
contains(Object elem)	Retorna un valor booleano si el elemento se encuentra en la lista (*true* si esta y *false* si no).
get(int pos)	Retorna el elemento de la posición indicada o se ejecuta la excepción *`IndexOutOfBoundsException`* si la posición está fuera del número de elementos que contiene la lista
indexOf(Object elem)	Retorna la posición donde encontró la primera coincidencia con el elemento que está buscando o un -1 en caso de no encontrarlo
isEmpty()	Retorna *true* si la lista está vacía o *false* si no lo está
lastIndexOf(Object elem)	Retorna la posición donde encontró la última coincidencia con el elemento que está buscando o un -1 en caso de no encontrarlo.

remove(int pos)	Retorna y elimina el elemento de la lista en la posición indicada o genera la excepción `IndexOutOfBoundsException` si la posición está fuera de la lista
revomeRange(int posIni, int posFin)	Elimina un rango de elementos de la lista desde la posición inicial hasta la posición final -1
set(int pos, Object elem)	Retorna y reemplaza el elemento que se encuentra en la posición indicada, por el elemento pasado como argumento, si la posición está fuera de los elementos de la lista ejecuta la excepción `IndexOutOfBoundsException`
size()	Retorna el número de elementos de la lista
trimToSize()	Reduce la capacidad de la lista al número de elementos actuales en la lista
ensureCapacity(int cap)	Aumenta la capacidad de la lista el número de posiciones indicada

(4) LINKEDLIST:

La clase LinkedList, permite crear una estructura de datos con nodos llamada lista ligada. El objeto creado a partir de la clase LinkedList es por lo tanto una estructura flexible que crece de forma dinámica. Esta estructura puede ser ideal para el manejo de pilas, colas y listas, y se encuentra definida en la librería `util`.

(a) Constructores de la clase LinkedList:

LinkedList lista=new LinkedList()	Crea un objeto llamado lista que es controlado por el inicio y el final de la misma
LinkedList lista=new LinkedList(Collection col)	Crea una lista con la colección de elementos que se pasan como argumento apoyándose de la clase `Collection`

(b) Métodos de la clase LinkedList:

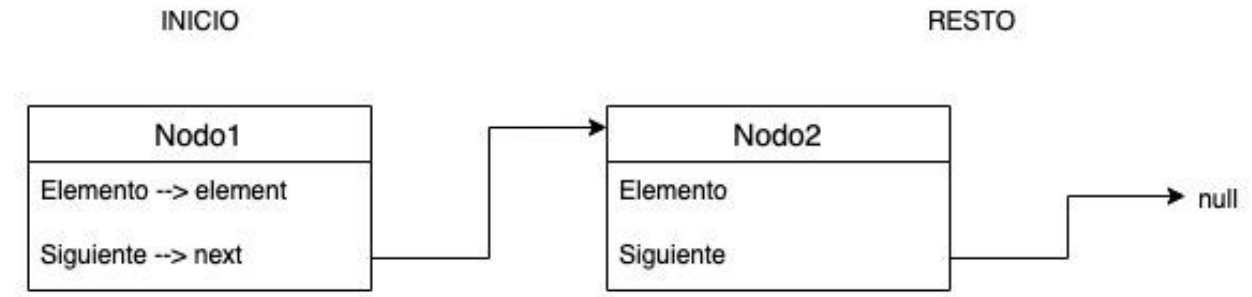
add(Object elem)	Inserta un elemento al final de la lista retornando un valor booleano si se pudo insertar o no
add(int pos, Object elem)	Inserta un elemento en la posición indicada desplazando los demás elementos una posición hacia el final de la lista, lanza la excepción `IndexOutOfBoundsException` si la posición está fuera del tamaño de la lista
addFirst(Object elem)	Inserta un elemento al principio de la lista
addLast(Object elem)	Inserta un elemento al final de la lista

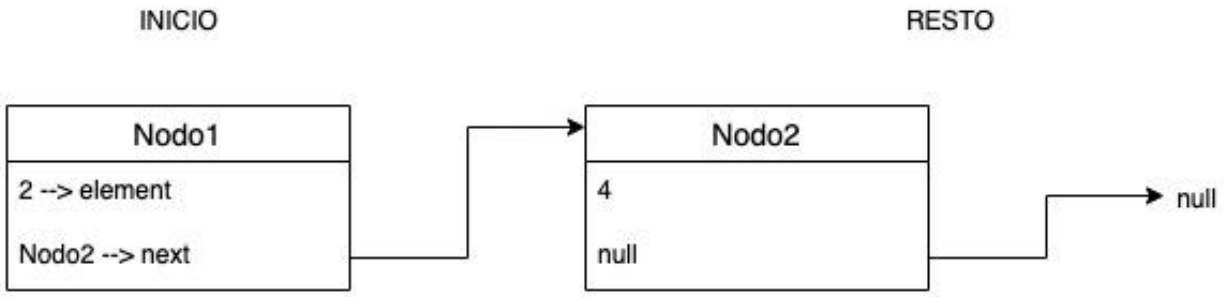
clean()	Elimina todos los elementos de la lista
clone()	Regresa una copia de la lista
contains(Object elem)	Retorna un valor booleano si el elemento se encuentra en la lista (*true* si esta y *false* si no).
get(int pos)	Retorna el elemento de la posición indicada o se ejecuta la excepción `IndexOutOfBoundsException` si la posición está fuera del número de elementos que contiene la lista
getFirst()	Retorna el primer elemento de la lista o genera la excepción `NoSuchElementException`, si la lista está vacía
getLast()	Retorna el último elemento de la lista o genera la excepción `NoSuchElementException`, si la lista está vacía
indexOf(Object elem)	Retorna la posición donde encontró la primera coincidencia con el elemento que está buscando o un -1 en caso de no encontrarlo
isEmpty()	Retorna *true* si la lista está vacía o *false* si no lo está
lastIndexOf(Object elem)	Retorna la posición donde encontró la última coincidencia con el elemento que está buscando o un -1 en caso de no encontrarlo
remove(int pos)	Retorna y elimina el elemento de la lista en la posición indicada o genera la excepción `IndexOutOfBoundsException` si la posición está fuera de la lista
removeFirst()	Retorna y elimina el primer elemento de la lista o genera la excepción `NoSuchElementException`, si la lista está vacía
revomeLast()	Retorna y elimina el último elemento de la lista o genera la excepción `NoSuchElementException`, si la lista está vacía
set(int pos, Object elem)	Retorna y reemplaza el elemento que se encuentra en la posición indicada, por el elemento pasado como argumento, si la posición está fuera de los elementos de la lista ejecuta la excepción `IndexOutOfBoundsException`
size()	Retorna el número de elementos de la lista

Ejercicio: implementar las interfaces dadas *ILista.java* e *IListaOps.java* y probar su correcto funcionamiento con la ayuda de los tester *ListaTest.java* y *ListaOpsTest.java*. Se proporciona también la clase *Node.java* con la implementación correspondiente a un nodo.

Ejercicio: proyecto de entrega ***ClubDelLector***

Explicación nodos visualmente con un ejemplo:






```
    if(inicio==null)
      return null;
    else{
    Node aux = inicio, anterior =null;
    while(aux!=null){
      anterior=aux;
      aux=aux.next;
    }
    return anterior.element
    }

Iteracion 1 -> aux = nodo1, anterior = null
                anterior = nodo1, aux=nodo2
Iteracion 2 -> aux = nodo2,
                anterior = nodo2, aux = null
Iteracion 3 -> aux = null -> PARO


-> RETORNO -> nodo2.element -> 4
```