Estructuras de datos
Clase práctica 4

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN

Contenido

- Listas enlazadas

Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Listas enlazadas

Para trabajar con listas, Java ya tiene implementada la clase *LinkedList* de la librería java.util.*

```
LinkedList<Integer> miLista= new LinkedList<Integer>();
```

También puede ser Byte, Short, Long, Float, Double (comenzando con mayúscula)

Además, los métodos que nos interesan son prácticamente los mismos que los de la clase *Vector*:

`addFirst(e)`	Ingresa el elemento *e* al inicio de la lista
`add(e)`	Ingresa el elemento *e* al final de la lista
`add(i,e)`	Ingresa el elemento *e* en la posición *i* de la lista
`clear()`	Borra todos los elementos

`get(i)`	Devuelve el elemento en la posición *i* de la lista (indexación)
`getFirst()`	Devuelve el primer elemento de la lista
`getLast()`	Devuelve el último elemento de la lista
`isEmpty()`	Devuelve verdadero si la lista está vacío
`indexOf(e)`	Devuelve la posición de la primera ocurrencia del elemento *e* dentro de la lista, o -1 si no está (búsqueda)
`lastIndexOf(e)`	Devuelve la posición de la última ocurrencia del elemento *e* dentro de la lista, o -1 si no está (búsqueda)
`remove(i)*`	Borra el elemento en la posición *i* de la lista (*i* es un *int*)
`removeFirst()`	Borra el primer elemento de la lista
`removeLast()`	Borra el último elemento de la lista
`remove(e)*`	Borra la primera ocurrencia del elemento *e* dentro de la lista. Devuelve *true* si el elemento se encuentra.
`set(i,e)`	Reemplaza el elemento en la posición *i* por *e*
`size()`	Devuelve la cantidad de elementos en la lista

*Cuidado, en caso que la lista guarde elementos de tipo entero, `remove(x)` entenderá que `x` es el índice, no el elemento. Si lo que se quiere es borrar en este caso es el elemento, se deberá usar: `miLista.remove(miLista.indexOf(x))`

Recorrido de una lista enlazada

Según los métodos presentados previamente, la forma de recorrer todos los elementos de una lista, para por ejemplo sumarlos, sería:

```
int s = 0;
for (int i = 0; i < miLista.size(); i++) {
   s += miLista.get(i);
}
```

Pero a partir de lo que vimos en clase ¿cuál sería la eficiencia de este procedimiento? $O(N^2)$

Una forma de hacerlo en *O(N)* sería usando un iterador:

```
int s = 0;
Iterator<Integer> miIterador = miLista.listIterator();
while(miIterador.hasNext()){
   s += miIterador.next();
}
```

Donde el tipo del iterador debe coincidir con el tipo de la lista. Pero cuidado, al hacer `miIterador = miLista.listIterator()` le estamos diciendo que cree un iterador para el estado actual de la lista. Si luego dicho estado cambia (se agregan o se borran elementos) y se desea usar de nuevo el iterador, habrá que "actualizarlo".

Otra utilidad del iterador es borrar de la lista el elemento en el que el iterador está "parado". Para esto se usa el método remove():

```
Iterator<Integer> miIterador = miLista.listIterator();
while(miIterador.hasNext()){
    miIterador.remove();
}
```

Ejemplo:

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        LinkedList<Integer> miLista = new LinkedList<Integer>();
        Iterator<Integer> miIterador;
        miLista.add(1);
        miLista.add(2);
        miLista.add(4);
        miLista.add(8);
        miIterador = miLista.listIterator();
        while(miIterador.hasNext())
            System.out.println(miIterador.next());
        miLista.add(16);
        miIterador = miLista.listIterator();
        while(miIterador.hasNext())
            System.out.println(miIterador.next());
    }
}
```