Listas Enlazadas

Estructuras de Datos

1

Diferentes Tipos

- **Simple**
- **De doble extremo**
- **Ordenada**
- **Doblemente enlazada**

2

Limitaciones: Elemento previo

- **En numerosas ocasiones, nos encontramos con la imposibilidad de acceder al elemento anterior**
  - **Una lista de dos extremos y eliminar el último elemento**
  - **No se puede buscar desde los dos extremos**

- **Una Lista doblemente enlazada resuelve este problema**
  - **Permite recorrerla hacia atrás y hacia adelante**
  - **Cada enlace contiene referencias al anterior y al siguiente**
    - **¡Excelente!**
  - **Pero recuerde, siempre hay una desventaja**
    - **Memoria**
    - **Además, los algoritmos sencillos se vuelven más lentos**

3

Nuestra nueva clase Link

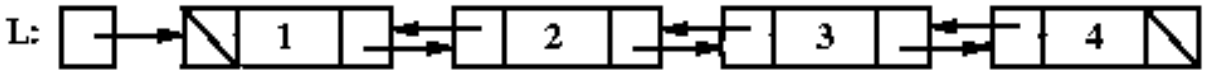
```
class Link
{
  public int iData;
  public Link previous;
  public Link next;
}
```

- **next and iData estaban antes, previous es la nueva referencia**
  - **Para el primer elemento, previous es null**
  - **Para el último elemento, next es null**

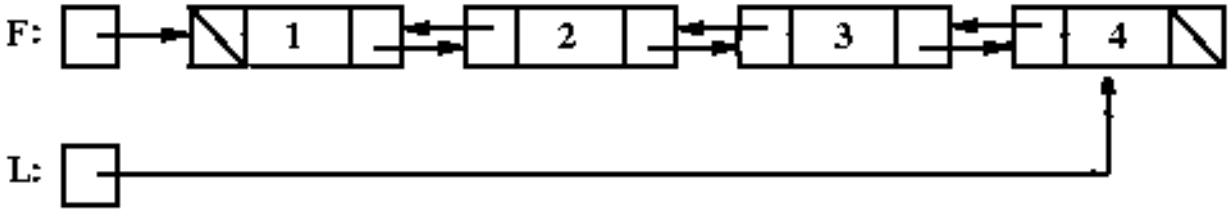
4

Lista Doblemente Enlazada…

- Un solo extremo (‘L’ referencia a la lista)



- Doble extremo
  - ‘F’ referencia al Primer elemento y ‘L’ referencia al último



5

Recorrido Inverso - O(n)

- El Recorrido hacia delante es igual al visto antes
  - Utilice current para hacer referencia a un enlace (Link), y en repetidas asignelo a current.next
- El Recorrido hacia atrás es nuevo
  - Esto sólo se puede hacer convenientemente si la lista es de dos extremos
  - Ahora repetidamente le asignamos a current la referencia a current.previous

6

Implementación Java

- **Métodos**
  - **isEmpty(), comprueba si está vacía**
    insertFirst (), insertar al principio
    insertLast (), insertar al final
    InsertDespues (), inserte en el medio
    deleteFirst (), elimine al inicio
    deleteLast (), borrar al final
    DeleteKey (), elimine en el medio
    displayForward (), recorrido hacia adelante
    displayBackward (), recorrido hacia atrás

7

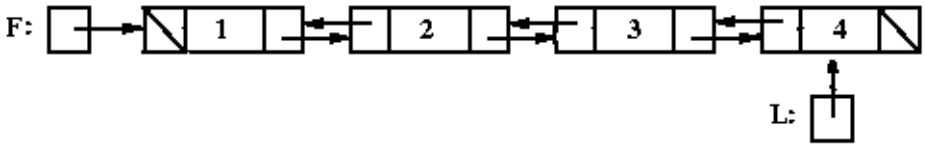
isEmpty() - O(1)

- **La función más simple**
- **Retorna true si first es null.**

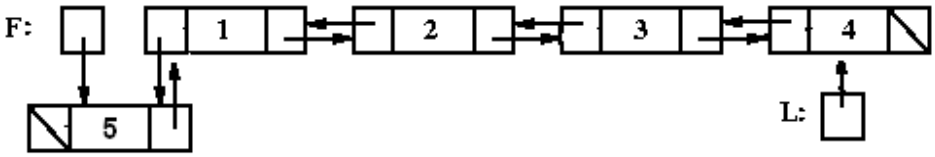
8

insertFirst() - O(1)

- **Pasos**
  - **Crear un nuevo enlace (Link)**
  - **Establezca su referencia next a first**
  - **Ajuste la primera referencia a previous al nuevo enlace**
  - **Asigne a first (y a last si la lista antes estaba vacía) la referencia al nuevo enlace**
- **Antes**



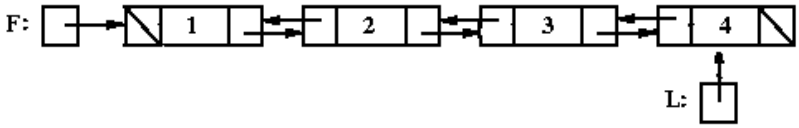
- **Después**



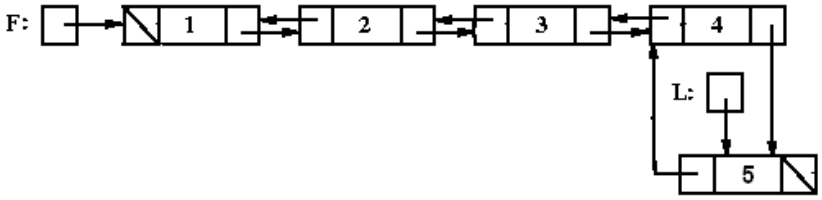
9

insertLast() - O(1)

- **Pasos**
  - **Crear un nuevo enlace (Link)**
  - **Asigne a la referencia de previous al último**
  - **Asigne a la última referencia de next al nuevo enlace (Link)**
  - **Asigne al último (y a first si antes la lista estaba vacía) para hacer referencia al nuevo enlace**
- **Antes**



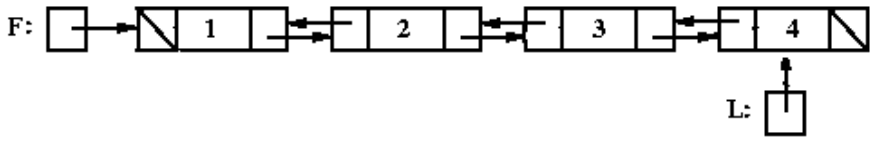
- **Después**



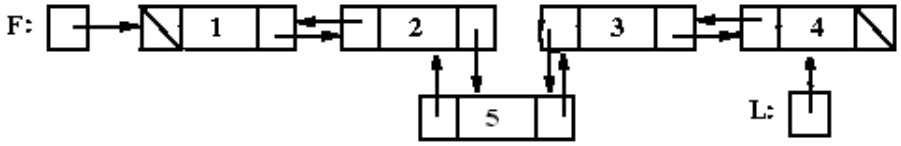
10

insertDespues() - O(n)

- **Pasos**
  - **Encuentra el elemento en lista desde el cual puedas insertar después de él (current)**
  - **Asigne a la referencia previous de current.next al nuevo enlace (link)**
  - **Asigne a la referencia next del enlace con current.next**
  - **Asigne current.next al nuevo enlace**
  - **Asigne a la referencia previous del enlace con current**
- **Antes**



- **Después**

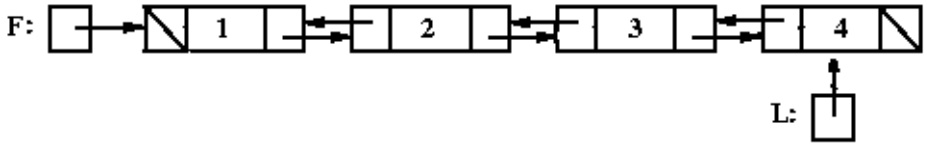


11

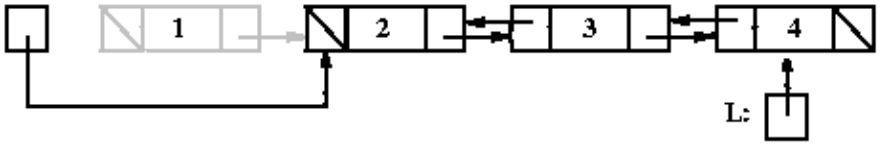
deleteFirst() - O(1)

- **Pasos**
  - **Asigne a la referencia previa a first.next con null**
    - **Recuerde que first.next podria ser null!!**
  - **Asigne first con first.next**

- **Antes**



- **Después**

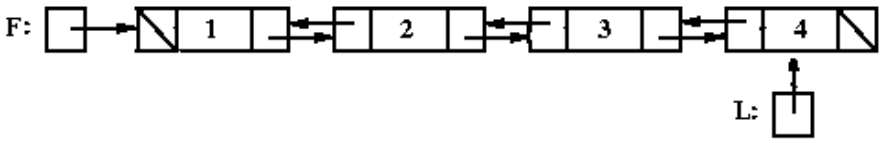


12

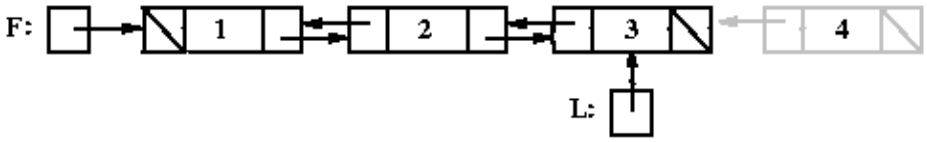
deleteLast() - O(1)

- **Pasos**
  - **Asigne a la referencia next de last.previous con null**
    - **Recuerde que last.previous podria ser null!!**
  - **Asigne a last con last.previous**

- **Antes**



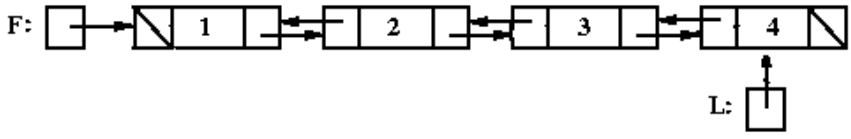
- **Después**



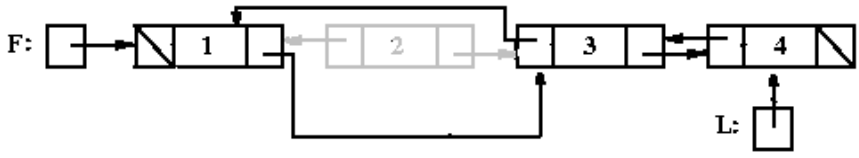
13

deleteKey() - O(n)

- **Pasos**
  - **Encuentra la clave, llámela current**
  - **Asigne la referencia current.previous de next con current.next**
  - **Asigne a la referencia previous de current.next con current.previous**
    - **Asegúrate de manejar el caso cuando cualquiera de ellos es null!! Esto seria equivalente a deleteFirst() o deleteLast()**
- **Antes**



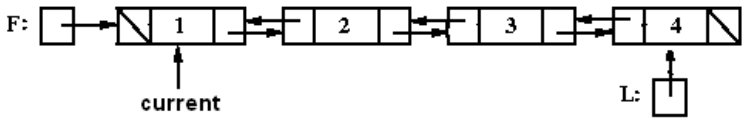
- **Después**



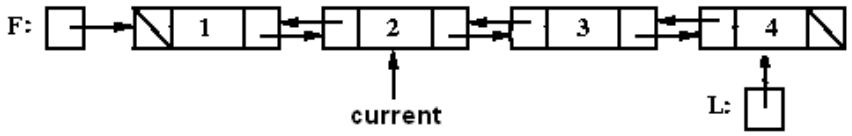
14

displayForward() - O(n)

- Use una referencia *current* para iterar a través de los elementos
  - Inicialmente es igual a *first*, imprima el valor
  - Asigne a *current* con *current.next*
- Pare cuando *current* sea null
- Antes de asignar *current* con *current.next*:



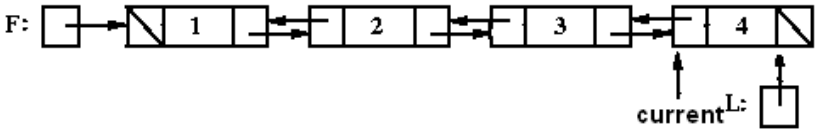
- Después de asignar *current* con *current.next*:



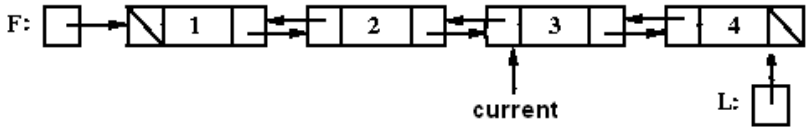
15

displayBackward() O(n)

- Use una referencia current para iterar a través de los elementos
  - Inicialmente es igual a *last*, imprima el valor
  - Asigne a *current* con *current.previous*
- Pare cuando *current* sea null
- Antes de asignar a *current* con *current.previous*:



- Después de asignar a *current* con *current.previous*:



16