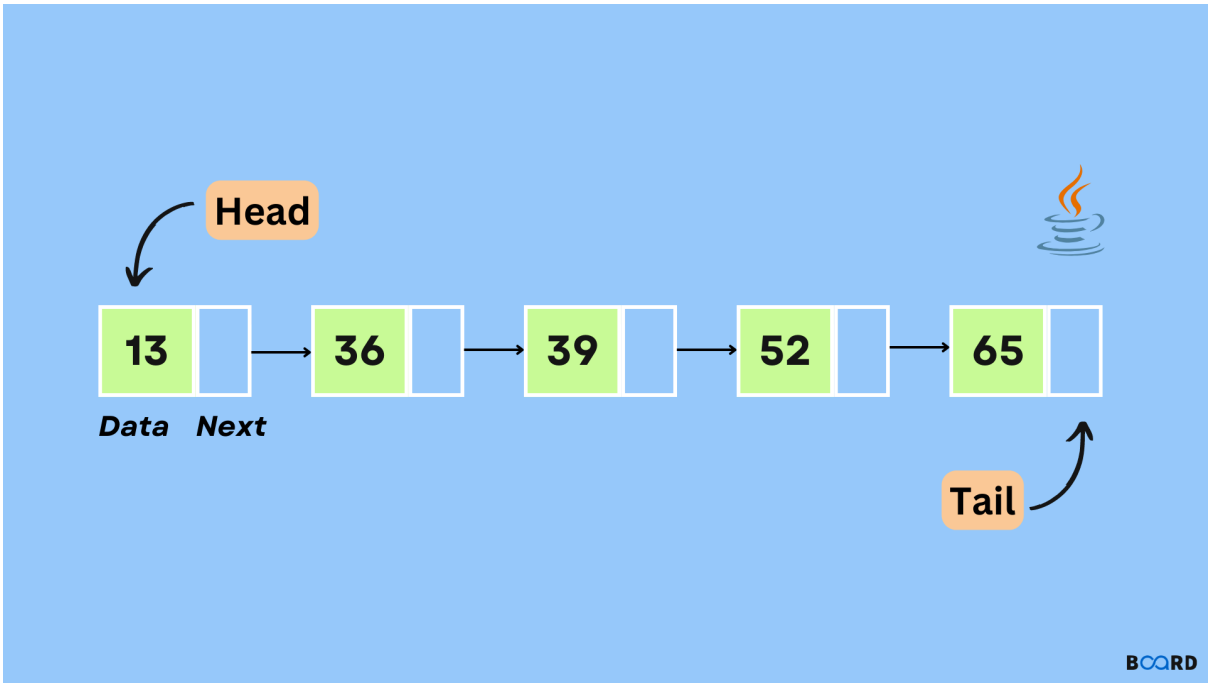
LINKEDLIST



1ºDAM
Programación
Simón Chica Suárez

Índice:

INTRODUCCIÓN

- Vamos a ver en profundidad la clase LinkedList, mostrando ejemplos y explicaciones de sus métodos más significativos. Los ejemplos prácticos serán mostrados en eclipse.

¿QUÉ ES LINKEDLIST?

- La clase `LinkedList` de Java forma parte del Marco de Colecciones de Java e implementa las interfaces `List` y `Deque`. En Java es parte del paquete `java.util` y permite almacenar elementos en una lista enlazada. Se puede utilizar para representar estructuras de datos como colas y pilas debido a su eficiencia en la inserción y eliminación de elementos.

¿CUÁNDO LA UTILIZO?

- `LinkedList` se utiliza cuando se necesitan inserciones y supresiones frecuentes. Es especialmente útil cuando no se conoce de antemano el tamaño de la lista o cuando hay que añadir o eliminar elementos en ambos extremos de la lista.

CARACTERÍSTICAS

- Implementa las interfaces `List` y `Deque`, permitiendo el acceso como lista y como una cola doblemente terminada.
- Permite elementos duplicados y mantiene el orden de inserción.
- Es eficiente para insertar y eliminar elementos en cualquier posición, pero tiene acceso secuencial en lugar de aleatorio.

SINTAXIS DEL LINKEDLIST

Para declarar y crear una lista enlazada en Java, se usa la siguiente sintaxis:

```
LinkedList<Type> linkedList = new LinkedList<>();
```

- `LinkedList<Type>` define una lista enlazada que almacena elementos del tipo `Type`.
- `new LinkedList<>()` crea una nueva instancia de la lista.
- `Type` es un tipo de dato genérico que se debe especificar al instanciar la lista, como `Integer`, `String`, `Double`...

MÉTODOS DEL LINKEDLIST(LOS MÁS IMPORTANTES)

```
void                add(int index, E element)
```

El método `add(int index, E element)` inserta un elemento en la posición especificada de la `LinkedList`, desplazando los elementos siguientes una posición hacia la derecha.

```
boolean             add(E e)
```

El método `add(E e)` de `LinkedList` en Java agrega un elemento al final de la lista y devuelve `true` si la operación fue exitosa.

```
void                addFirst(E e)
void                addLast(E e)
```

- **`void addFirst(E e)`** → Agrega un elemento al inicio de la `LinkedList`, moviendo los demás elementos una posición hacia la derecha.

`void addLast(E e)` → Agrega un elemento al final de la `LinkedList`, similar a `add(E e)`.

```
void                clear()
```

El método **`void clear()`** elimina todos los elementos de la `LinkedList`, dejándola vacía.

```
boolean             contains(Object o)
```

El método **`boolean contains(Object o)`** verifica si la `LinkedList` contiene un elemento específico y devuelve `true` si lo encuentra, o `false` en caso contrario.

```
E                   get(int index)
E                   getFirst()
E                   getLast()
```

`E get(int index)` → Devuelve el elemento en la posición especificada de la `LinkedList`.

`E getFirst()` → Retorna el primer elemento de la `LinkedList`.

`E getLast()` → Retorna el último elemento de la `LinkedList`.

E	removeFirst()
boolean	removeFirstOccurrence(Object o)
E	removeLast()

removeFirst() → Elimina y devuelve el primer elemento de la `LinkedList`.

removeLast() → Elimina y devuelve el último elemento de la `LinkedList`.

boolean	offerFirst(E e)
boolean	offerLast(E e)

boolean offerFirst(E e) → Inserta un elemento al inicio de la `LinkedList` y devuelve `true` si la operación fue exitosa.

boolean offerLast(E e) → Inserta un elemento al final de la `LinkedList` y devuelve `true` si la operación fue exitosa.

EJEMPLOS BÁSICOS

Ejemplo 1: Operaciones básicas

```java
import java.util.LinkedList;

public class LinkedListExample {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();

        // Adding elements
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Accessing elements
        System.out.println("First element: " + list.getFirst());
        System.out.println("Last element: " + list.getLast());

        // Removing elements
        list.removeFirst();
        list.removeLast();

        System.out.println("List after removals: " + list);
    }
}
```

Ejemplo 2: Iterar a través de una lista enlazada

```
import java.util.LinkedList;
import java.util.Iterator;

public class LinkedListIteration {
    public static void main(String[] args) {
        LinkedList<Integer> numbers = new LinkedList<>();
        numbers.add(10);
        numbers.add(20);
        numbers.add(30);

        Iterator<Integer> iterator = numbers.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

Ejemplo 3: Utilizar LinkedList como cola

```
import java.util.LinkedList;
import java.util.Queue;

public class LinkedListAsQueue {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();

        // Enqueue elements
        queue.add("First");
        queue.add("Second");
        queue.add("Third");

        // Dequeue elements
        System.out.println("Removed: " + queue.poll());
        System.out.println("Queue after removal: " + queue);
    }
}
```

Webgrafía

https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?punto=73&codigo=152&inicio=60

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/LinkedList.html

https://jcodepoint.com/java/ordenar-un-map/