SEGUNDO EXAMEN PARCIAL - LISTAS ENLAZADAS EN JAVA

Solorzano Rodas, Julio Andres

UDVD ED1

Explicación del Código

El código implementa una **lista enlazada simple** en Java utilizando memoria dinámica. Esto significa que los nodos se almacenan en el heap y se gestionan mediante referencias. La clase Node representa cada nodo, mientras que la clase LinkedList proporciona métodos para manipular la lista.

Flujo de Búsqueda de un Nodo (Método contains)

Cuando se busca un valor en la lista:

1. Se empieza en la cabeza (head).
2. Se compara data del nodo actual con el valor buscado.
3. Si hay coincidencia, retorna true.
4. Si no, se avanza al siguiente nodo hasta encontrarlo o llegar a null.
5. Si no se encuentra, retorna false.

Ejemplo de búsqueda de 4:

- Compara 8 (no coincide) → avanza.
- Compara 6 (no coincide) → avanza.
- Compara 4 (coincide) → retorna true.

Ejemplo de búsqueda de 8:

- Compara 8 (coincide) → retorna true.

Descripción de Clases y Métodos

Clase Node

Representa un nodo en la lista enlazada.

- int data: Almacena el valor del nodo.
- Node next: Referencia al siguiente nodo en la lista.

Clase LinkedList

Implementa la estructura de la lista enlazada y sus operaciones:

Métodos Implementados

- void add(int data): Agrega un nodo al final de la lista.
- void addFirst(int data): Agrega un nodo al inicio de la lista.
- void addMiddle(int data, int position): Agrega un nodo en una posición específica.
- void remove(int data): Elimina el nodo que contiene el valor especificado.
- void printList(): Imprime la lista enlazada.
- void reverse(): Invierte el orden de la lista.
- boolean contains(int value): Verifica si un valor está en la lista.

Clase Main

Contiene el método main que ejecuta el programa paso a paso, realizando operaciones sobre la lista enlazada:

1. Agrega nodos (inicio, medio y final).
2. Imprime la lista después de cada operación.
3. Elimina un nodo y muestra la lista actualizada.
4. Verifica si ciertos valores existen en la lista.
5. Invierte la lista y la muestra nuevamente.
6. Agrega un nodo adicional y muestra la lista final.

Salida Esperada

El programa muestra en pantalla el resultado de cada operación, permitiendo visualizar cómo la lista enlazada cambia a lo largo de su ejecución.

Explicación de la Clase Node y su Propósito

La clase Node representa un elemento dentro de la lista enlazada. Cada nodo tiene:

- Un **atributo data** para almacenar un valor entero.
- Un **atributo next** que es una referencia a otro Node, permitiendo la conexión entre ellos.

Proposito

Cada `Node` permite construir una estructura dinámica donde los nodos se enlazan entre sí para formar una **lista enlazada**.

Explicación de Cada Método

La clase LinkedList maneja los nodos y permite agregar, eliminar, recorrer y modificar la lista.

🔹 **Método add(int data)**

Agrega un nodo al final de la lista.

- Si la lista está vacía, head se convierte en el nuevo nodo.
- Si no, se recorre hasta el último nodo y se agrega el nuevo al final.

✅ **Ejemplo:** add(5)
Lista antes: null
Lista después: 5 -> null

🔹 **Método addFirst(int data)**

Agrega un nodo al inicio de la lista.

- Se crea un nuevo nodo que apunta al nodo que antes era la cabeza.
- Ahora la cabeza es el nuevo nodo.

✅ **Ejemplo:** addFirst(3)
Lista antes: 5 -> null
Lista después: 3 -> 5 -> null

🔹 **Método addMiddle(int data, int position)**

Agrega un nodo en una posición específica.

✅ **Ejemplo:** addMiddle(7, 1)
Lista antes: 3 -> 5 -> null
Lista después: 3 -> 7 -> 5 -> null

🔹 **Método remove(int data)**

Elimina un nodo con un valor específico.

✅ **Ejemplo:** remove(7)
Lista antes: 3 -> 7 -> 5 -> null
Lista después: 3 -> 5 -> null

🔹 **Método printList()**

Imprime la lista enlazada.

✅ **Ejemplo:**
Salida: 3 -> 5 -> null

🔹 **Método reverse()**

Revierte la lista enlazada.

✅ **Ejemplo:** reverse()
Lista antes: 3 -> 5 -> null
Lista después: 5 -> 3 -> null

🔹 **Método contains(int value)**

Verifica si un valor está en la lista.

✅ **Ejemplo:** contains(3)
Salida: true

✅ **Ejemplo:** contains(7)
Salida: false

Explicación de la Memoria Dinámica y Referencias

- Cada nodo se crea dinámicamente en memoria (heap).
- Las referencias (next) permiten enlazar nodos.
- Cuando se elimina un nodo, su referencia se pierde y Java lo libera (garbage collection).

Conclusión

- Este programa demuestra cómo:
  - ✅ Manejar listas enlazadas en Java.
  - ✅ Agregar, eliminar y buscar nodos.
  - ✅ Usar memoria dinámica correctamente.
  - ✅ Implementar métodos fundamentales de listas enlazadas.