Universidad Da Vinci De Guatemala

Facultad de Ingeniería

Carrera: Ingeniería en Sistemas

“Segundo examen parcial”

Fernando Sáenz Aguilar

Carnet: 202402168

Curso: Estructura de Datos

Guatemala, abril de 2025

Introducción

Este proyecto implementa una lista enlazada simple en Java. El objetivo es demostrar la comprensión de memoria dinámica, uso de referencias, y la implementación de métodos fundamentales para manipular listas enlazadas.

Una lista enlazada es una estructura de datos lineal donde cada elemento (nodo) contiene datos y una referencia al siguiente nodo en la secuencia. Esta implementación incluye todas las operaciones básicas necesarias para su manipulación.

Explicación de la clase Node y su propósito

La clase Node es el componente fundamental de la lista enlazada. Representa un nodo individual dentro de la estructura de datos y tiene dos propósitos principales:

- Almacenar datos: Cada nodo contiene un valor (en este caso un entero en el atributo data).
- Mantener la referencia al siguiente nodo: El atributo next es una referencia que apunta al siguiente nodo en la lista, creando así la "cadena" o "enlace" que conecta todos los elementos.

Estructura del Proyecto

El proyecto está organizado en dos clases principales:

- **Node**: Representa un nodo individual en la lista enlazada, que contiene un valor entero y una referencia al siguiente nodo.
- **LinkedList**: Implementa la lista enlazada y sus operaciones.

Métodos Implementados

La clase LinkedList implementa los siguientes métodos:

1. **add(int data):** Agrega un nuevo nodo al final de la lista.
2. **addFirst(int data):** Agrega un nuevo nodo al inicio de la lista.
3. **addMiddle(int data, int position):** Agrega un nuevo nodo en una posición específica.
4. **remove(int data):** Elimina el nodo que contiene el valor especificado.
5. **printList():** Imprime todos los valores de la lista.
6. **reverse():** Invierte el orden de los nodos en la lista.
7. **contains(int value):** Verifica si un valor existe en la lista.

Explicación del manejo de memoria dinámica y referencias

Memoria Dinámica

1. **Creación dinámica de nodos**: Cada vez que usamos new Node(data), estamos solicitando memoria en tiempo de ejecución. A diferencia de un array que tiene tamaño fijo predefinido, la lista enlazada puede crecer o reducirse según sea necesario.
2. **Espacio bajo demanda**: Solo ocupamos la memoria que realmente necesitamos, ni más ni menos.
3. **Liberación automática**: En Java, cuando un nodo deja de ser referenciado (por ejemplo, cuando lo eliminamos de la lista), el recolector de basura se encarga de liberar esa memoria.

Referencias

1. **Enlaces entre nodos**: La referencia next es el "pegamento" que une toda la estructura. Cada nodo sabe quién es el siguiente en la cadena.
2. **La cabeza (head)**: Es una referencia especial que apunta al primer nodo de la lista. Si perdemos esta referencia, perdemos acceso a toda la lista.
3. **Referencias temporales**: Cuando recorremos o modificamos la lista, usamos referencias temporales como current que nos permiten movernos por la lista sin perder la referencia a la cabeza.

Conceptos Demostrados

Este proyecto demuestra los siguientes conceptos de programación:

- Manejo de memoria dinámica
- Manipulación de referencias en Java
- Implementación de estructuras de datos básicas
- Algoritmos para manipulación de listas enlazadas
- Manejo de casos especiales (lista vacía, inserción en posiciones límite)

Cómo Ejecutar el Código

Para ejecutar este proyecto, necesitas tener instalado Java Development Kit (JDK) en tu sistema.

1. Clona este repositorio.
2. Navega hasta el directorio del proyecto.
3. Compila el Código.
4. Ejecuta el programa.

Ejemplo de Uso

El método main en la clase LinkedList proporciona un ejemplo completo de uso siguiendo la secuencia de operaciones específica para el Grupo 1 (estudiantes con carnet terminado en 0, 2, 4, 6, 8):

1. Agrega el número 0 al inicio de la lista
2. Agrega el número 2 al final de la lista
3. Agrega el número 4 en la posición 1
4. Muestra la lista actual
5. Elimina el número 2
6. Muestra la lista después de la eliminación
7. Agrega el número 6 al final
8. Verifica si el número 4 está en la lista
9. Verifica si el número 8 está en la lista
10. Revierte la lista
11. Muestra la lista después de revertirla
12. Agrega el número 8 al inicio
13. Muestra la lista final

Ejemplo de Salida

```
=== Operaciones para Grupo 1 ===
1. Agregando el número 0 al inicio de la lista
2. Agregando el número 2 al final de la lista
3. Agregando el número 4 al medio de la lista (en la posición 1)
4. Lista actual:
0 -> 4 -> 2 -> null
5. Removiendo el número 2 de la lista
6. Lista después de la eliminación:
0 -> 4 -> null
7. Agregando el número 6 al final de la lista
8. ¿La lista contiene el número 4? true
9. ¿La lista contiene el número 8? false
10. Revirtiendo la lista
11. Lista después de revertir:
6 -> 4 -> 0 -> null
12. Agregando el número 8 al inicio de la lista
13. Lista final:
8 -> 6 -> 4 -> 0 -> null
```