PONTIFÍCA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

ESCOLA POLITÉCNICA

CURSO DE ENGENHARIA DE SOFTWARE

MATHEUS ZUCCHI GIORDANI

EXERCÍCIO 2 PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. MARCO AURELIO SOUZA MANGAN

PORTO ALEGRE

2025

Introdução

Este relatório técnico tem como objetivo apresentar um estudo aprofundado da classe LinkedList do Java Collections Framework . O trabalho foca na demonstração prática de uso da LinkedList através de um exemplo de código, e na explicação sobre o seu funcionamento e funcionalidades disponíveis.

Exemplo de uso do LinkedList:

A LinkedList em Java é uma implementação da interface List e Deque , que utiliza uma estrutura de dados de lista duplamente encadeada. Isso significa que cada elemento (nó) na lista contém uma referência para o elemento anterior e para o próximo elemento. Essa característica confere à LinkedList vantagens específicas em operações de inserção e remoção no início ou no fim da lista, bem como em qualquer posição, em comparação com estruturas como ArrayList .

O código java usado como exemplo mostra as principais funções do uso da LinkedList:

```java
import java.util.LinkedList;

public class LinkedListPoo {

   public static void main(String[] args) {
      // Criação da LinkedList
      LinkedList<String> lista = new LinkedList<>();
      System.out.println("Lista inicial: " + lista);

      // Operações de Inserção
      System.out.println("\nInserção");
      lista.addFirst("Elemento A");
      System.out.println("Após addFirst do Elemento A" + lista);

      lista.addLast("Elemento C");
      System.out.println("Após addLast do Elemento C"+ lista);

      lista.add(1, "Elemento B");
      System.out.println("Após add do Elemento B" + lista);

      lista.add("Elemento D");
      System.out.println("Após add do Elemento D" + lista);
```

```
        // Operação de percorrer
        System.out.println("\nPercorrer");
        System.out.print("Percorrendo a lista: ");
        for (String item : lista) {
            System.out.print(item + ", ");
        }
        System.out.println();

        // Operação de Pesquisa
        System.out.println("\nPesquisa");
        String elementoProcurado = "Elemento B";
        int indice = lista.indexOf(elementoProcurado);
        if (indice != -1) {
            System.out.println(elementoProcurado + " encontrado no índice: " + indice);
            System.out.println("Elemento no índice 0: " + lista.get(0));
        } else {
            System.out.println(elementoProcurado + " não encontrado.");
        }

        // Operações de Remoção
        System.out.println("\nRemoção");
        lista.removeFirst();
        System.out.println("Após removeFirst(): " + lista);

        lista.removeLast();
        System.out.println("Após removeLast(): " + lista);

        lista.remove("Elemento B");
        System.out.println("Após remove do Elemento B: " + lista);

        if (!lista.isEmpty()) {
            lista.remove(0);
            System.out.println("Após remove(0): " + lista);
        }

        System.out.println("\nLista final: " + lista);
    }
}
```

Desenvolvimento:

O texto fala sobre como usar uma LinkedList que guarda textos String. A classe principal LinkedListPoo é o ponto de partida do programa. Embora a lista seja declarada como LinkedList, ela segue as regras de uma List, um conceito que chamamos de **polimorfismo** (o objeto tem "várias formas" de se comportar).

1. Adicionar Itens

Os métodos addFirst, addLast, add(índice, elemento) e add(elemento) mostram como colocar itens no início, no fim ou em um lugar específico da lista.

Vantagem: A LinkedList é muito rápida para adicionar itens. Ela só precisa mudar os ponteiros dos itens vizinhos, sem ter que empurrar todos os outros itens para o lado.

2. Andar e Olhar a Lista

Andar pela lista é feito com o for-each. Isso é um jeito fácil de ler e rápido de escrever para olhar cada item.

3. Encontrar Itens

Para achar um item, usamos o método indexOf() para encontrar a primeira vez que ele aparece.

O método get(índice) deixa você pegar um item em um lugar específico.

4. Tirar Itens (Remoção)

A LinkedList é flexível para tirar itens com métodos como removeFirst, removeLast, remove(Objeto) e remove(índice).

Ótimo desempenho: Tirar o item do começo ou do fim é muito rápido (assim como a inserção).

Pode ser lento: Remover um item pelo seu valor ou pela sua posição no meio da lista exige encontrar o item primeiro, o que pode ser uma operação com custo linear.

O método isEmpty() é um atalho útil que verifica se a lista tem itens. Isso evita erros que poderiam acontecer se você tentasse operar em uma lista vazia.

5. Conceitos de Programação

Neste exemplo, você não vê classes **herdando** outras ou implementando **interfaces** de forma clara no código.

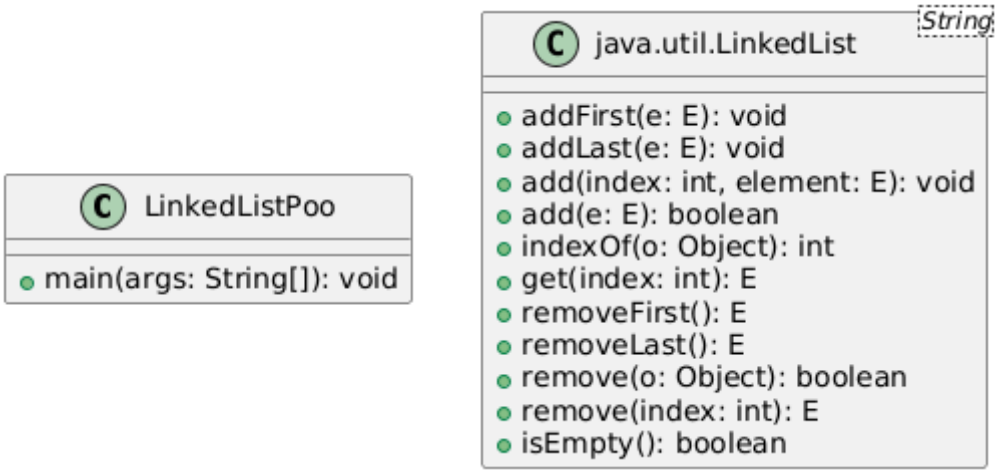
No entanto, a própria LinkedList já implementa várias interfaces (como List). Isso significa que ela promete ter um conjunto de regras e comportamentos que essas interfaces definem.

Sobrescrita e **Sobrecarga** não são mostrados, mas são básicos no funcionamento interno da LinkedList e do Java.

Em resumo: A LinkedList é ideal quando você precisa adicionar ou tirar itens rapidamente nas pontas. Ela é menos eficiente quando você precisa pesquisar ou acessar um item no meio.

Diagrama UML:

O diagrama UML a seguir representa as classes envolvidas no exemplo, LinkedListPoo e LinkedList , e suas relações. A LinkedListPoo usa a LinkedList para suas operações.



Questão 1

(ENADE 2017 - Adaptada) Considere as seguintes afirmações sobre as estruturas de dados ArrayList e LinkedList em Java:

I. A ArrayList é mais eficiente para operações de inserção e remoção no meio da lista, devido à sua estrutura baseada em array dinâmico.

II. A LinkedList é mais eficiente para operações de acesso a elementos por índice (get(index)), pois cada elemento armazena referências para o próximo e o anterior.

III. A LinkedList consome mais memória que a ArrayList para armazenar o mesmo número de elementos, devido ao overhead de armazenamento das referências de próximo e anterior em cada nó.

IV. Ambas ArrayList e LinkedList implementam a interface List do Java Collections Framework.

Estão corretas apenas as afirmações:

a) I e II

b) III e IV

c) I, II e III

d) II, III e IV

e) I, III e IV

Resposta: b) III e IV

Justificativa:

I. Incorreta: A ArrayList é ineficiente para inserções e remoções no meio, pois exige o deslocamento de todos os elementos subsequentes. A LinkedList é mais eficiente para essas operações.

II. Incorreta: O acesso por índice em LinkedList é O(n) no pior caso, pois requer a travessia da lista. Em ArrayList, o acesso por índice é O(1).

III. Correta: Cada nó da LinkedList armazena o dado e duas referências (próximo e anterior), resultando em maior consumo de memória por elemento em comparação com a ArrayList, que armazena apenas o dado.
IV. Correta: Ambas as classes são implementações da interface List, o que lhes permite serem usadas de forma polimórfica onde uma List é esperada.

Questão 2

(Concurso Público - Adaptada) Analise o trecho de código Java a seguir:

```
LinkedList<Integer> numeros = new LinkedList<>();
numeros.add(10);
numeros.addFirst(5);
numeros.addLast(20);
numeros.add(1, 15);
numeros.remove(Integer.valueOf(10));
System.out.println(numeros.get(0));
```

Qual será a saída impressa no console após a execução deste código?
a) 5
b) 15
c) 20
d) 10
e) Erro de compilação

Resposta: a) 5

Justificativa:
LinkedList<Integer> numeros = new LinkedList<>(); -> []
numeros.add(10); -> [10]
numeros.addFirst(5); -> [5, 10]
numeros.addLast(20); -> [5, 10, 20]
numeros.add(1, 15); -> [5, 15, 10, 20] (Insere 15 no índice 1)
numeros.remove(Integer.valueOf(10)); -> [5, 15, 20] (Remove a primeira ocorrência do objeto 10)
System.out.println(numeros.get(0)); -> Acessa o elemento no índice 0, que é 5.

Aprendizados e Dificuldades Resolvidas

Durante a elaboração deste trabalho, foi possível aprofundar o entendimento sobre as características da LinkedList , especialmente suas vantagens e desvantagens em relação a outras implementações da interface List , como a ArrayList . A principal dificuldade foi entender as diferenças de performance no uso de outra implementação da interface List

sobre a LinkedList. A prática de elaboração do relatório utilizando os termos de POO reforçou a aplicação desses conceitos em um contexto prático.

Conclusão

A LinkedList é uma estrutura de dados poderosa e flexível em Java, ideal para cenários onde inserções e remoções frequentes são necessárias, especialmente nas extremidades da lista. Compreender suas características, desempenho e consumo de memória é fundamental para a escolha da estrutura de dados mais adequada em diferentes contextos de programação.

Referencias

Oracle. **Java Platform, Standard Edition (SE) 21 API Specification**. Disponível em: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/LinkedList.html. Acesso em: 7 out. 2025.

PLANTUML. **PlantUML**. Disponível em: https://plantuml.com/. Acesso em: 7 out. 2025.

GEMINI. In: GOOGLE. **Gemini: modelo de linguagem**. Mountain View: Google, c2025. Disponível em: https://gemini.google.com/app/ . Acesso em: 7 out. 2025.

DEITEL, Paul J.; DEITEL, Harvey M.; FURMANKIEWICZ, Edson. **Java: como programar**. 8. ed. São Paulo: Pearson Prentice Hall, 2010. Cap. 20, p. 640, 642-647. Acesso em: 7 out. 2025.

HORSTMANN, Cay S.; CORNELL, Gary. **Core Java**. 8. ed. Rio de Janeiro: Prentice Hall Brasil, 2008. v. 1, cap. 13, p. 315-317. Acesso em: 7 out. 2025.

LINK UML:

www.plantuml.com/plantuml/png/TP11IWD144NtVOeYcqnGvW2DI3Pn8ICHN8gBodGvLEpk2jqrGP4SoBL7oCMCISC4Csx_-__Nf-PPCMaVl6azvWmDnpTo3MTz585F0m1m1G4vLfYwRE594yVk-MLcOIlipCwKv6IDMwnxPLyF9JSd--q4yBpx_c0jvLDus7-fECcmELXRtY1dw3F-Ab6WtNyxxiIM9UZS7QUi5LbOb0q3Ql0_Kd5qz6w1eru3UGeKTTHNqYiHJnWBE8QNhvLOMAxMrEhis5HeHtfMVq2B0X85sT9fxuNSu9WwdBWOKGoZbpWlmaO_gZ-XATHr3GJIQv8CeNpQxCoSekk3_m40