

Lista Encadeada em Java

1. Classe No

A classe No representa um único elemento (nó) da lista encadeada. Cada nó contém:

- Um valor inteiro (`valor`) para armazenar o dado.
- Um ponteiro (`proximo`) que aponta para o próximo nó na lista.

```
class No {  
    int valor;  
    No proximo;  
  
    No(int valor) {  
        this.valor = valor;  
        this.proximo = null;  
    }  
}
```

Construtor:

Ao criar um nó, inicializamos o valor e configuramos o ponteiro `proximo` como `null`, indicando que ele não aponta para nenhum outro nó inicialmente.

2. Classe ListaEncadeada

A classe ListaEncadeada gerencia os nós, fornecendo métodos para realizar as operações na lista.

Atributo:

- `No inicio`: O ponteiro que indica o primeiro nó da lista.

Construtor:

```
public ListaEncadeada() {  
    this.inicio = null;  
}
```

Inicializamos `inicio` como `null`, indicando que a lista está vazia.

2.1 Inserção de Elementos

1. Inserção no início

- Cria um novo nó.
- Faz o `proximo` do novo nó apontar para o atual início da lista.
- Atualiza `inicio` para o novo nó.

```
public void inserirInicio(int valor) {  
    No novoNo = new No(valor);  
    novoNo.proximo = inicio;  
    inicio = novoNo;  
}
```

2. Inserção no final

- Se a lista está vazia, o novo nó se torna o primeiro.
- Caso contrário, percorremos até o último nó e fazemos seu `proximo` apontar para o novo nó.

```
public void inserirFinal(int valor) {  
    No novoNo = new No(valor);  
    if (inicio == null) {  
        inicio = novoNo;  
    } else {  
        No atual = inicio;  
        while (atual.proximo != null) {  
            atual = atual.proximo;  
        }  
        atual.proximo = novoNo;  
    }  
}
```

3. Inserção em uma posição específica

- Verifica se a posição é válida.
- Se a posição for 0, chama `inserirInicio`.

- Percorre até a posição anterior e ajusta os ponteiros.

```
public void inserirPosicao(int valor, int posicao) {
    if (posicao < 0) { ... }
    if (posicao == 0) { ... }
    No novoNo = new No(valor);
    No atual = inicio;
    for (int i = 0; i < posicao - 1; i++) {
        if (atual == null) { ... }
        atual = atual.proximo;
    }
    novoNo.proximo = atual.proximo;
    atual.proximo = novoNo;
}
```

2.2 Remoção de Elementos

1. Remoção do início

- Apenas atualiza o ponteiro `inicio` para o próximo nó.

```
public void removerInicio() {
    if (inicio == null) {
        System.out.println("Lista vazia!");
        return;
    }
    inicio = inicio.proximo;
}
```

2. Remoção do final

- Percorre até o penúltimo nó e ajusta seu `proximo` para `null`.

```
public void removerFinal() {
    if (inicio == null) { ... }
    if (inicio.proximo == null) {
        inicio = null;
        return;
    }
}
```

```

    }
    No atual = inicio;
    while (atual.proximo.proximo != null) {
        atual = atual.proximo;
    }
    atual.proximo = null;
}

```

3. Remoção de uma posição específica

- Verifica se a posição é válida.
- Percorre até o nó anterior à posição e ajusta os ponteiros.

```

public void removerPosicao(int posicao) {
    if (posicao < 0 || inicio == null) { ... }
    if (posicao == 0) { ... }
    No atual = inicio;
    for (int i = 0; i < posicao - 1; i++) { ... }
    atual.proximo = atual.proximo.proximo;
}

```

2.3 Pesquisa

Busca o valor na lista e retorna a posição. Se não encontrar, retorna -1.

```

public int pesquisar(int valor) {
    No atual = inicio;
    int posicao = 0;
    while (atual != null) {
        if (atual.valor == valor) {
            return posicao;
        }
        atual = atual.proximo;
        posicao++;
    }
    return -1;
}

```

2.4 Atualização

Atualiza o valor de um nó em uma posição específica.

```
public void atualizar(int posicao, int novoValor) {
    if (posicao < 0) { ... }
    No atual = inicio;
    for (int i = 0; i < posicao; i++) { ... }
    atual.valor = novoValor;
}
```

2.5 Exibição

Imprime os elementos da lista até chegar ao final.

```
public void exibir() {
    if (inicio == null) {
        System.out.println("Lista vazia!");
        return;
    }
    No atual = inicio;
    while (atual != null) {
        System.out.print(atual.valor + " -> ");
        atual = atual.proximo;
    }
    System.out.println("null");
}
```

3. Classe Principal (Main)

A classe principal gerencia o programa, oferecendo um menu interativo para o usuário escolher operações.

1. Menu de opções

- Exibe as opções disponíveis.
- Lê a entrada do usuário e chama o método correspondente na lista.

2. Entrada e validação

- Recebe valores e posições do usuário, validando os dados.
- Exibe mensagens de erro ou confirmação.

```
public class Main {
    public static void main(String[] args) {
        ListaEncadeada lista = new ListaEncadeada();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n--- Menu ---");
            System.out.println("1. Inserir no início");
            // Demais opções...
            int opcao = scanner.nextInt();
            switch (opcao) {
                case 1: // Implementar ações
                case 9: lista.exibir(); break;
                case 0:
                    System.out.println("Saindo...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Opção inválida!");
            }
        }
    }
}
```

1. A lista encadeada é gerenciada pela classe ListaEncadeada, que oferece métodos para manipular os nós.
2. A interação com o programa é feita via um menu na classe principal.
3. Cada método possui validações para garantir o uso correto