

# Tipos de Dados - Java

## Primitivos

INTEIROS	byte (8 bits)
	short (16 bits)
	int (32 bits)
	long (64 bits)
PONTO FLUTUANTE	float (32 bits)
	double (64 bits)
CARACTER	char, 16 bits, sem sinal
BOOLEANO	boolean, true / false

# Tipos de Dados - Java

Não Primitivos

CLASSES

STRING

ARRAY / VETOR

MÉTODOS

ATRIBUTOS

# Problema: Lista de Supermercado

## Requisitos

Precisamos construir uma aplicação para registrarmos itens que estão em falta e precisam ser comprados quando irmos ao supermercado. Principais operações:

- 1 - Adicionar item
- 2 - Listar itens presentes na lista
- 3 - Remover item da lista

# Problema: Lista de Supermercado

Requisitos

Precisamos construir uma aplicação para registrarmos itens que estão em falta e precisam ser comprados quando irmos ao supermercado. Principais operações:

- 1 - Adicionar item
- 2 - Listar itens presentes na lista
- 3 - Remover item da lista

Hora de codificar!

```
package implementation;

public interface Supermarket { 3 usages 1 implementation

    void add(final String item); 1 usage 1 implementation

    void print(); 1 usage 1 implementation

    void delete(final int index); 1 usage 1 implementation

}
```

Interface Supermarket

```

package implementation;

import java.sql.SQLOutput;

public class SupermarketArrray implements Supermarket{ 2 usages

    private final String[] itens; 6 usages

    private int lastIndex; 9 usages

    public SupermarketArrray(final int size) { 1 usage
        itens = new String[size];
        lastIndex = -1;
    }

    @Override 1 usage
    public void add(String item) {
        if(lastIndex == itens.length-1){
            System.err.println("Lista de Supermercado cheia");
        }else{
            lastIndex++;
            itens[lastIndex] = item;
        }
    }

    @Override 1 usage
    public void print() {
        System.out.println("#####");
        if (lastIndex < 0) {
            System.out.println("Lista de supermercado *vazia*");
        }
        System.out.println("#####");
        for (int i = 0; i <= lastIndex; i++){
            System.out.println(i + " = " + itens[i]);
        }
    }
}

```

Classe Supermarket

```
}
```

```
@Override 1 usage
```

```
public void delete(int index) {
```

```
    if (index >= 0 && index <= lastIndex){
```

```
        shift(index);
```

```
        lastIndex--;
```

```
    }else {
```

```
        System.err.println("Índice inválido: " + index);
```

```
    }
```

```
}
```

```
private void shift(int index) { 1 usage
```

```
    for (int i = index; i < lastIndex; i++){
```

```
        itens[i] = itens[i+1];
```

```
    }
```

```
}
```

```
}
```

Classe Supermarket

```
package main;

import implementation.Supermarket;
import implementation.SupermarketArrray;

import java.util.Scanner;

public class Main {

    private final static int SIZE = 3; 1 usage

    public static void main(String[] args) {
```

Classe Main

```
        Scanner scanner = new Scanner(System.in);
        Supermarket supermarket = new SupermarketArrray(SIZE);

        int opcao;
        do {
            System.out.println("\nLista de Compras ");
            System.out.println("1 - inserir ");
            System.out.println("2 - Listar ");
            System.out.println("3 - Remover");
            System.out.println("4 - Sair ");
            System.out.println("Escolha uma opção: ");
            opcao = scanner.nextInt();

            switch (opcao){
                case 1:
                    System.out.println("Digite o item a ser inserido");
                    String item = scanner.next();
                    supermarket.add(item);
                    break;
                case 2:
                    supermarket.print();
                    break;
                case 3:
                    System.out.println("Digite a posição do item a ser removido: ");
                    int index = scanner.nextInt();
                    supermarket.delete(index);
                    break;
```



```

43     case 4:
44         System.out.println("Saindo do programa...");
45         break;
46     default:
47         System.out.println("opção Inválida. Por favor escolha Novamente...");
48
49     }
50
51     }while (opcao !=4);
52
53     scanner.close();
54
55 }
56
57

```

Classe Main

Lista de Compras

- 1 - inserir
- 2 - Listar
- 3 - Remover
- 4 - Sair

Escolha uma opção:

1

Digite o item a ser inserido

feijao

Lista de Compras

- 1 - inserir
- 2 - Listar
- 3 - Remover
- 4 - Sair

Escolha uma opção:

2

#####

#####

0 - feijao

# Problema: Lista de Supermercado

Uso de Vetor

## Vantagens:

- 1 - Acesso direto aos elementos;
- 2 - Implementação simples;
- 3 - Uso eficiente de memória (alocação).

## Desvantagens:

- 1 - Tamanho fixo;
- 2 - Ineficiência em remoções no meio;
- 3 - Subutilização de memória.

# Estruturas de Dados

## Conceitos

Estrutura de Dados é uma forma específica de armazenar dados.

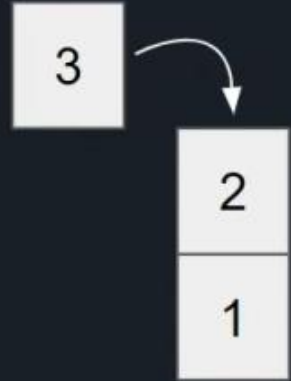
Tipo Abstrato de Dados (TAD) é o uso de uma estrutura de dados com operações ou comportamento específico.

# Estruturas de Dados

## Classificação



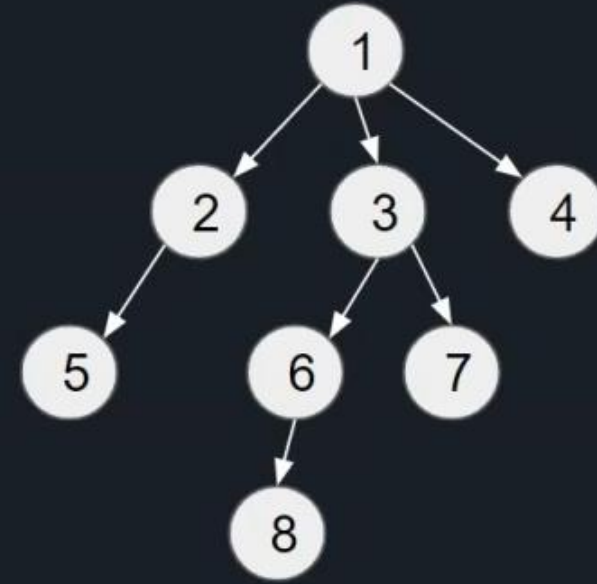
Lista ligada



Pilha



Fila



Árvores

## Conclusão

Em diversos cenários as estruturas nativas presentes nas linguagens de programação não são suficientes para a resolução de problemas de forma eficiente.

Solução: Estruturas de Dados

Como desenvolvedor de software, devemos conhecer algumas estruturas de dados, que possuem características mais flexíveis e dinâmicas, e possam ser aplicadas na resolução eficiente de problemas.

# Lista Ligada

## Conceito

Lista Ligada ou Lista Encadeada é uma estrutura de dados dinâmica, linear, formada por **nós**. Cada nó é capaz de armazenar uma informação e referenciar o próximo nó.

**Adicionar o valor 3**





# Lista Ligada

## Principais Operações

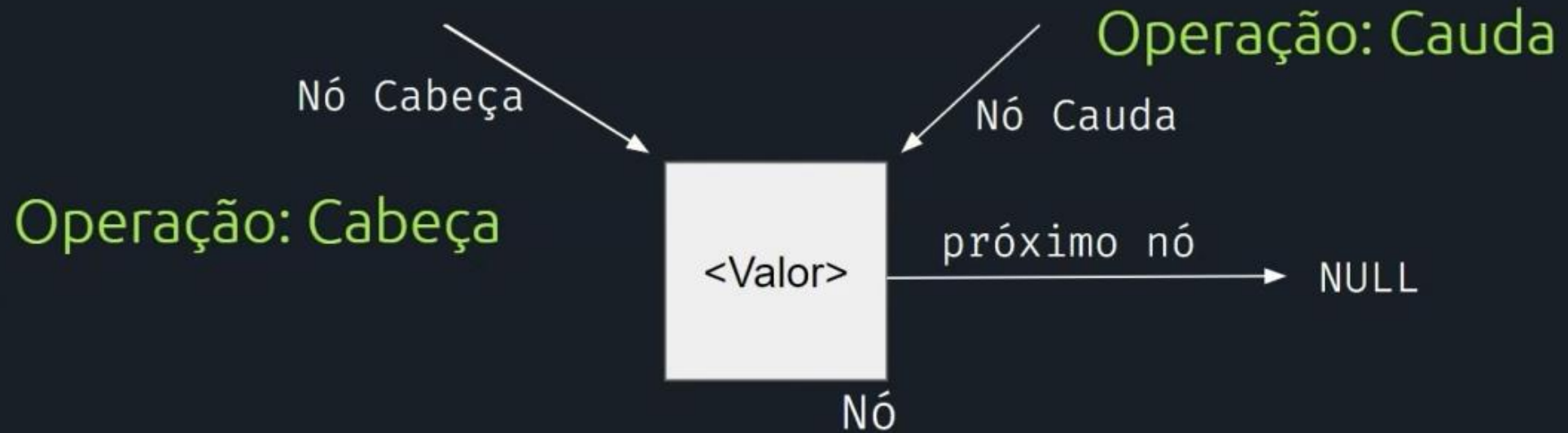
### Principais operações:

- 1 - Adicionar item
  - a) No início (*prepend*)
  - b) No fim (*append*)
  - c) Em uma determinada posição (*insert*)
- 2 - Ler item
  - a) Do início (*getHead*)
  - b) Do fim (*getTail*)
  - c) De uma determinada posição (*get*)
- 3 - Remover item
  - a) No início (*removeFirst*)
  - b) No fim (*removeLast*)
  - c) Em uma determinada posição (*delete*)
- 4 - Imprimir

### Operações “opcionais”:

- 1 - Tamanho atual (*getLength*)
- 2 - Está vazia? (*isEmpty*)
- 3 - Esvaziar lista (*makeEmpty*)

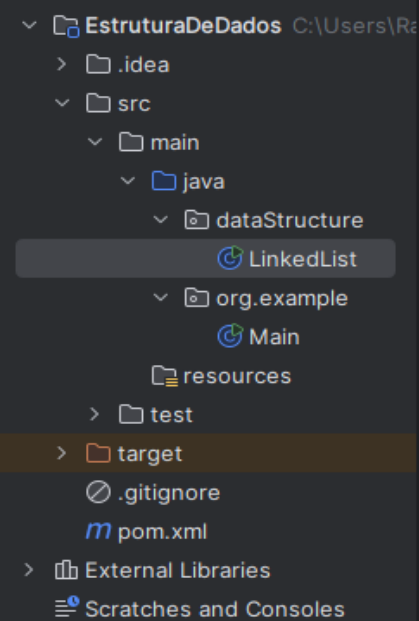
# Operação: Criar Lista e Esvaziar Lista



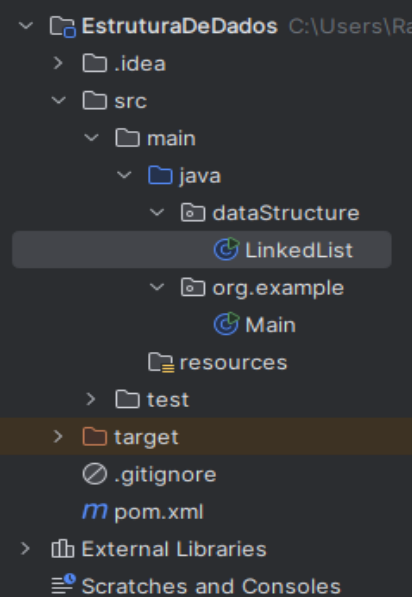
tamanho = 1

Operação: Tamanho





```
1 package dataStructure;
2
3 public class LinkedList {
4
5     private Node head; 5 usages
6
7     private Node tail; 3 usages
8
9     private int length; 3 usages
10
11
12
13
14     class Node{ 5 usages
15         String data; 3 usages
16
17         Node next; no usages
18
19         Node(String data) { 1 usage
20             this.data = data;
21         }
22     }
23
24
25     public LinkedList(String data){ 1 usage
26
27         length = 1;
28         Node newNode = new Node(data);
29         head = newNode;
30         tail = newNode;
31
32     }
33
34     public void getHead() { 1 usage
35         if (this.head == null){
36             System.out.println("Lista Vazia");
37         }else {
38             System.out.println("Head - " + head.data);
39         }
40     }
41 }
```



Run  LinkedList 



```
Process finished with exit code 0
```



# Operação: Imprimir



Saída:

Implementação do método:

```
public void print(){ 1 usage
```

```
    Node temp = this.head;
```

```
    while (temp != null){
```

```
        System.out.println(temp.data);
```

```
        temp = temp.next;
```

```
    }
```

```
}
```

Imprimindo o método:

```
public static void main(String[] args) {
```

```
    LinkedList list = new LinkedList( data: "elemento 1");
```

```
    list.getHead();
```

```
    list.getTail();
```

```
    list.getLength();
```

```
    System.out.println("#####");
```

```
    list.print();
```

```
    System.out.println("#####");
```

```
}
```

## Operação: Inserir no Final (append)



Tamanho = 2

## Operação: Inserir no Final (append)



Caso especial: `lista vazia`.

Tamanho = 3

Hora de codificar!

Implementação do método:

```
public void append(String data){ 2 usages

    Node newNode = new Node(data);
    if (length == 0){
        head = newNode;
        tail = newNode;
    }else{
        tail.next = newNode;
        tail = newNode;
    }
    length++;
}
```

Imprimindo o método:

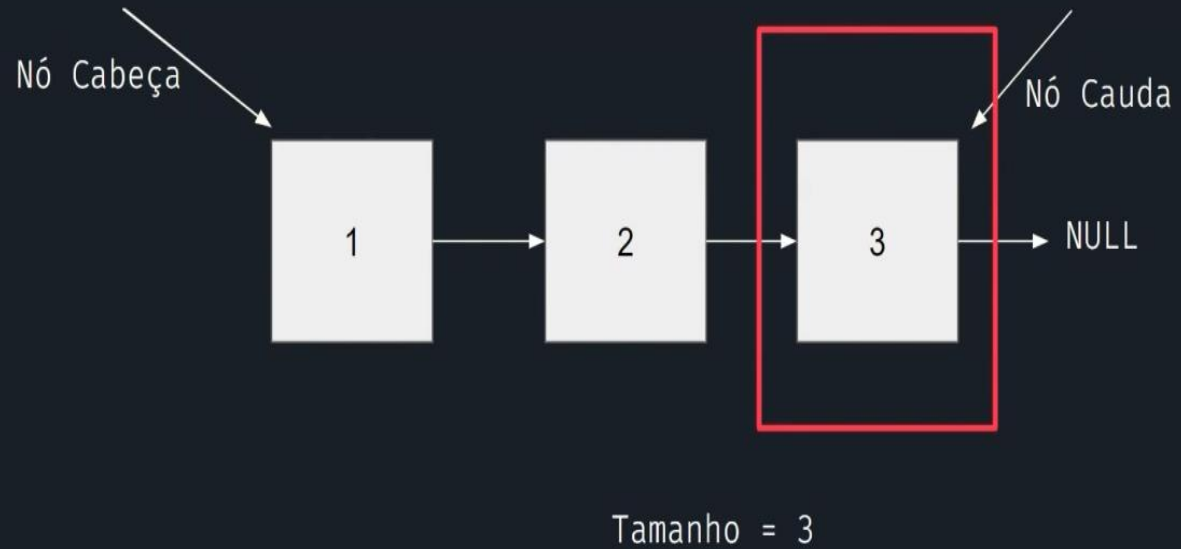
```
public static void main(String[] args) {

    LinkedList list = new LinkedList( data: "elemento 1");

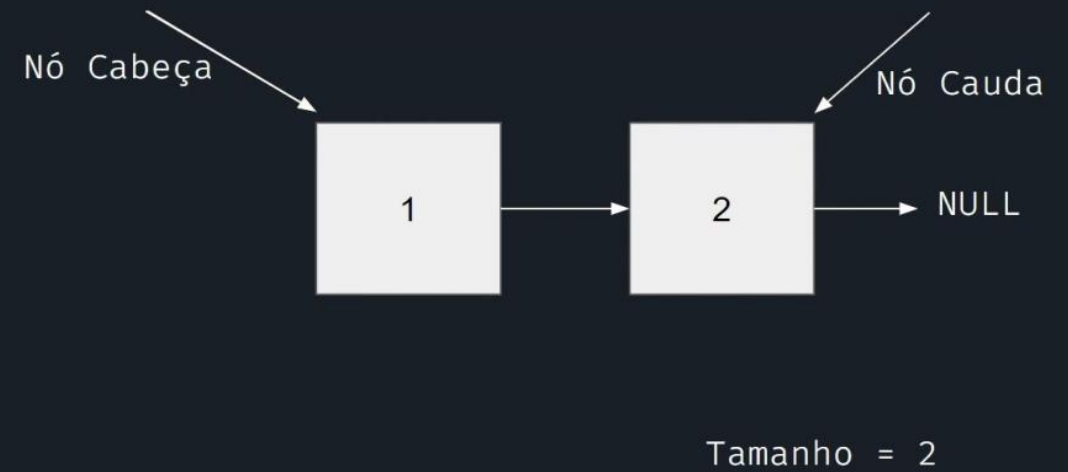
    list.append("elemento 2"); ←
    list.append("elemento 3"); ←

    list.getHead();
    list.getTail();
    list.getLength();
    System.out.println("#####");
    list.print();
    System.out.println("#####");
}
```

## Operação: Remover do Final



## Operação: Remover do Final



Caso especial: *lista vazia*.

Caso especial: *elemento único*.



## Implementação do método:

```
public Node removelast(){ 1 usage
    if (length == 0) return null;
    Node pre = head;
    Node temp = null;

    while (pre.next != tail) {
        pre = pre.next;
    }

    temp = tail;
    tail = pre;
    tail.next = null;

    length--;
    if (length == 0){
        head = null;
        tail = null;
    }

    return temp;
}
```

## Imprimindo o método:

```
public static void main(String[] args) {

    LinkedList list = new LinkedList( data: "elemento 1");

    list.append("elemento 2");
    list.append("elemento 3");

    System.out.println(list.removelast().data); ←
    //
    //     list.getHead();
    //     list.getTail();
    //     list.getLength();
    System.out.println("#####");
    list.print();
    System.out.println("#####");
}
```

# Operação: Inserir no início (*prepend*)



Tamanho = 2

Implementando o método:

```
public void prepend(String data) { 1 usage
    Node newNode = new Node(data);
    if (length == 0) {
        head = newNode;
        tail = newNode;
    } else {
        newNode.next = head;
        head = newNode;
    }
    length++;
}
```

Imprimindo o método:

```
public static void main(String[] args) {

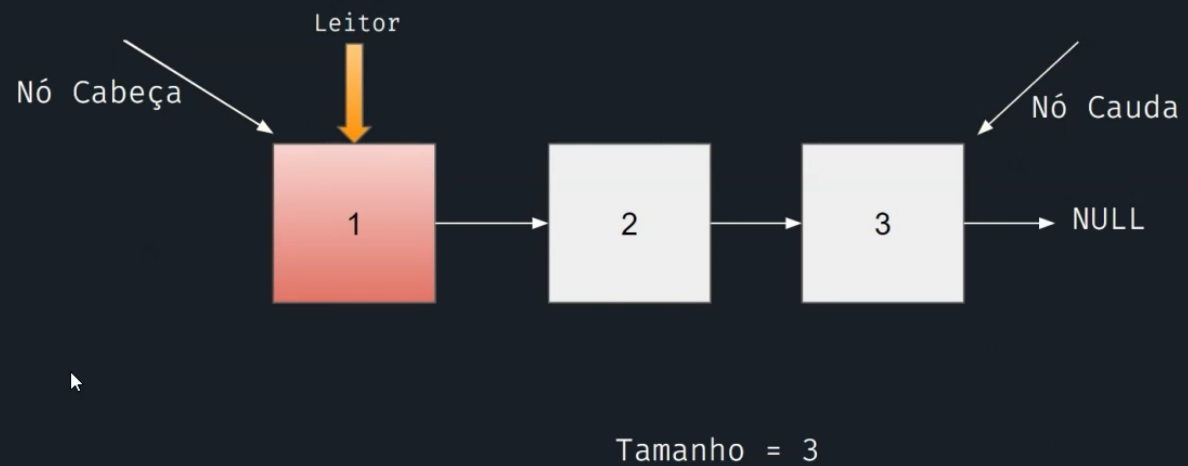
    LinkedList list = new LinkedList(data: "elemento 1");

    list.append("elemento 2");
    list.append("elemento 3");
    list.prepend(data: "elemento 0"); ←

    System.out.println(list.removeLast().data);

    list.getHead();
    list.getTail();
    list.getLength();
    System.out.println("#####");
    list.print();
    System.out.println("#####");
}
```

## Operação: Remover do início



## Operação: Remover do início



Caso especial: `lista vazia.`

Caso especial: `elemento único.`

Implementando o método:

```
public Node removeFirst(){ 1 usage
    if (length == 0) return null;
    Node temp = head;
    head = head.next;
    temp.next = null;
    length--;

    if (length == 0){
        head = null;
        tail = null;
    }
    return temp;
}
```

Imprimindo o método:

```
public static void main(String[] args) {

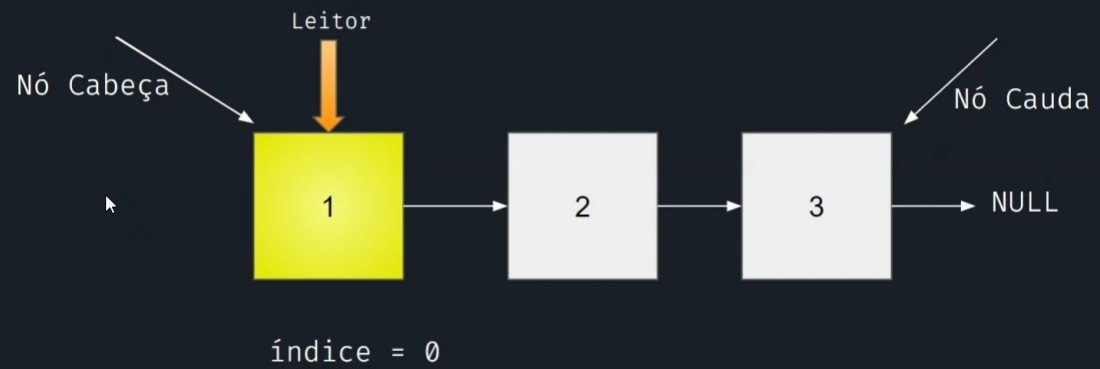
    LinkedList list = new LinkedList( data: "elemento 1");

    list.append("elemento 2");
    list.append("elemento 3");
    list.prepend( data: "elemento 0");

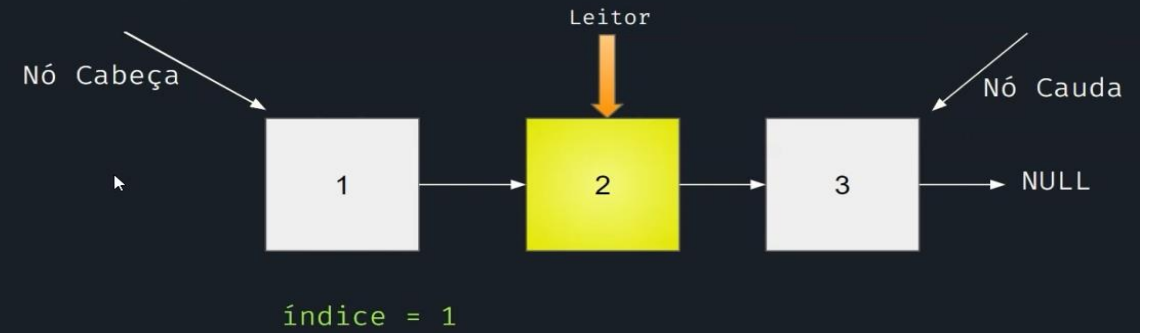
    System.out.println(list.removeFirst().data);

    list.getHead();
    list.getTail();
    list.getLength();
    System.out.println("#####");
    list.print();
    System.out.println("#####");
}
```

### Operação: Ler de um posição (*get*)

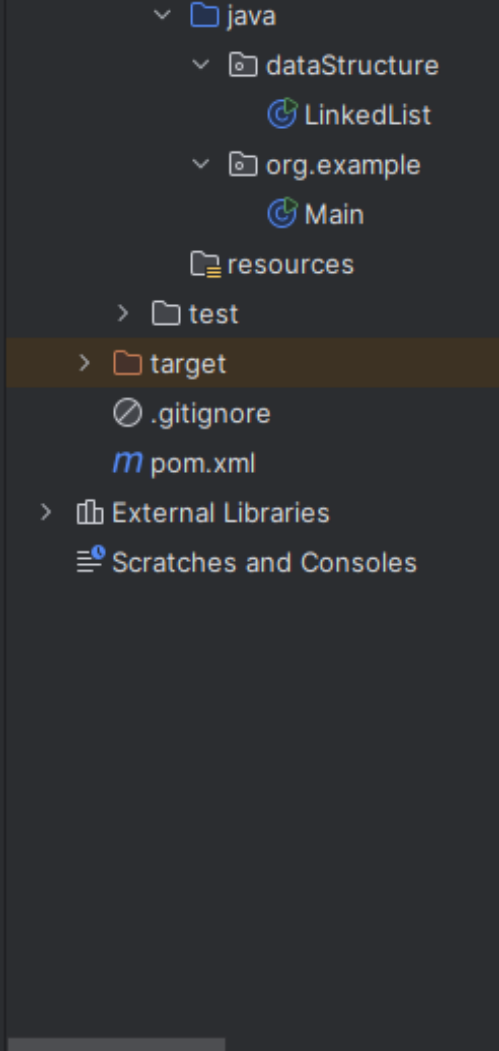


### Operação: Ler de um posição (*get*)



Importante: validar índice.

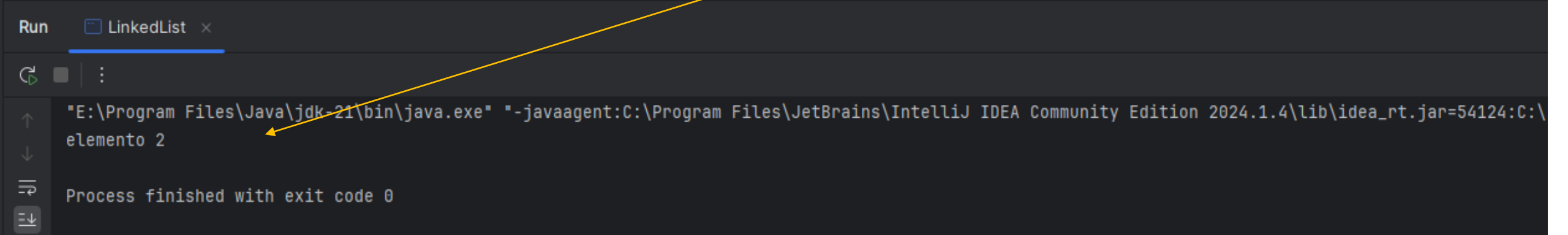
Fácil: alterar valor de uma posição



```
132 public Node get(int index){ 1 usage
133     if (index <0 || index >= length)return null;
134     Node temp = head;
135     for (int i = 0; i<index; i++){
136         temp =temp.next;
137     }
138     return temp;
139 }
140
141
142 public static void main(String[] args) {
143
144     LinkedList list = new LinkedList( data: "elemento 1");
145
146     list.append("elemento 2");
147     list.append("elemento 3");
148     list.prepend( data: "elemento 0");
149
150     System.out.println(list.get(2).data);
151
152 }
153
154 }
```

Implementando o método

Imprimindo o método



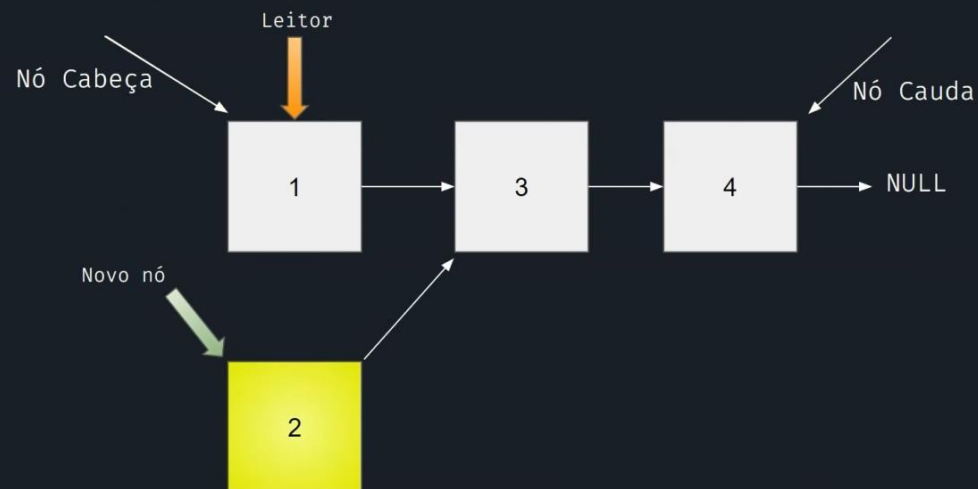
### Operação: Inserir em um posição específica



### Operação: Inserir em um posição específica



### Operação: Inserir em um posição específica



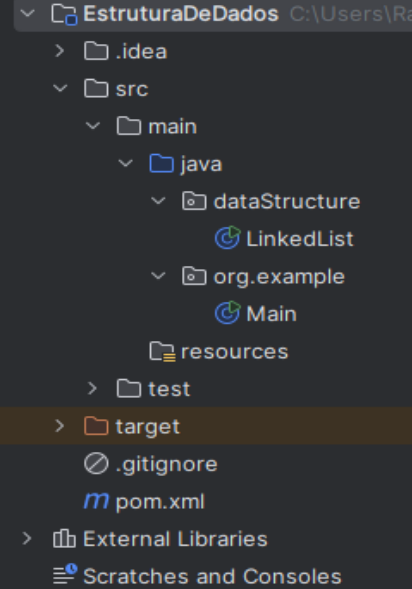
### Operação: Inserir em um posição específica



Importante: `validar índice.`

Casos especiais: `append`, `prepend`





```
public class LinkedList {  
    140  
    141     public boolean insert(int index, String data){ 1 usage  
    142         if (index < 0 || index > length) return false;  
    143         if (index == 0){  
    144             prepend(data);  
    145             return true;  
    146         }  
    147         if (index == length){  
    148             append(data);  
    149             return true;  
    150         }  
    151  
    152         Node newNode = new Node(data);  
    153         Node temp = get(index - 1);  
    154         newNode.next = temp.next;  
    155         temp.next = newNode;  
    156         length++;  
    157         return true;  
    158     }  
    159  
    160     public static void main(String[] args) {  
    161  
    162         LinkedList list = new LinkedList( data: "elemento 1");  
    163  
    164         list.append("elemento 2");  
    165         list.append("elemento 3");  
    166         list.prepend( data: "elemento 0");  
    167         list.insert( index: 3, data: "elemento 2,5");  
    168         list.print();  
    169     }  
    170 }
```

Implementando o método

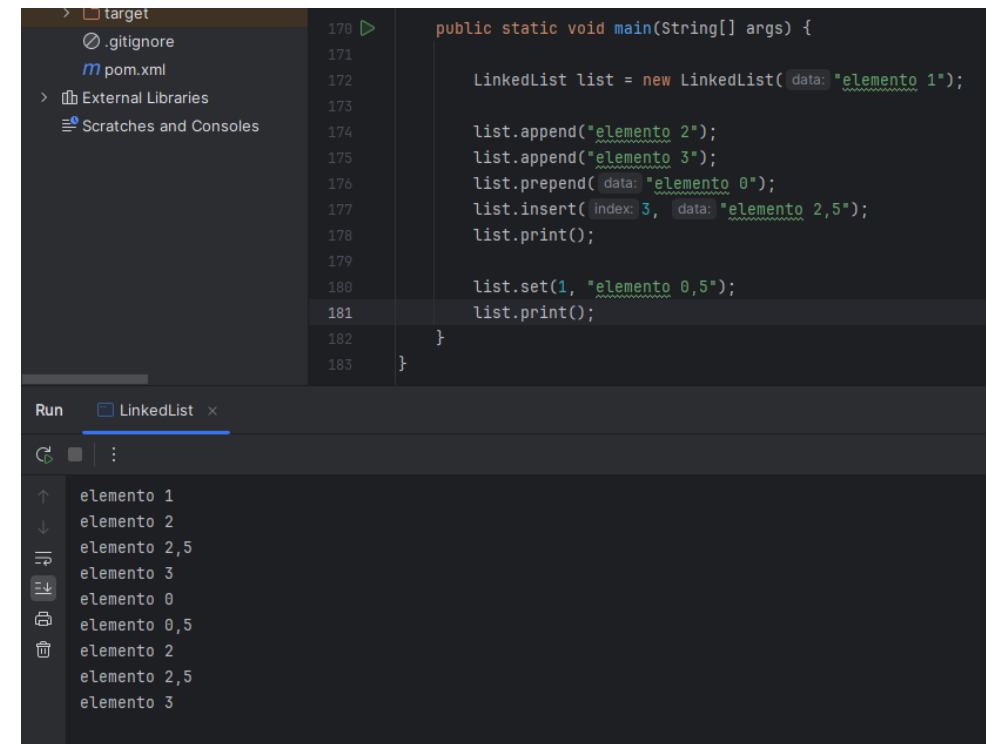
Imprimindo o método

Run LinkedList x

```
"E:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.4\lib\idea_rt.jar=54250:C:\Program Files\JetBrains\IntelliJ  
elemento 0  
elemento 1  
elemento 2  
elemento 2,5  
elemento 3
```

# Método set (Alterar o item)

```
public boolean set(int index, String data){ 1 usage
    Node temp = get(index);
    if (temp != null){
        temp.data = data;
        return true;
    }
    return false;
}
```



The screenshot shows an IDE with a project named 'target'. The file explorer on the left shows '.gitignore', 'pom.xml', 'External Libraries', and 'Scratches and Consoles'. The main editor displays the following Java code:

```
170 public static void main(String[] args) {
171     LinkedList list = new LinkedList(data: "elemento 1");
172
173     list.append("elemento 2");
174     list.append("elemento 3");
175     list.prepend(data: "elemento 0");
176     list.insert(index: 3, data: "elemento 2,5");
177     list.print();
178
179     list.set(1, "elemento 0,5");
180     list.print();
181 }
182
183 }
```

The Run console at the bottom shows the output of the program:

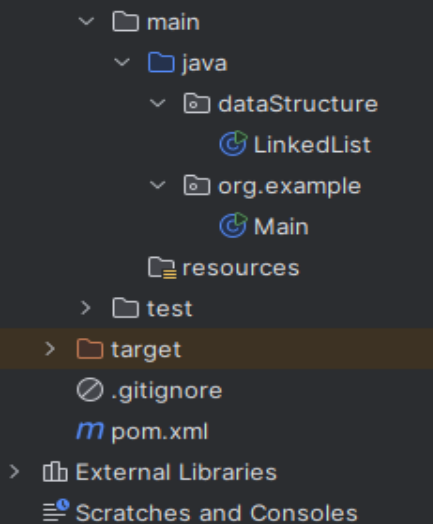
```
↑ elemento 1
↓ elemento 2
elemento 2,5
elemento 3
elemento 0
elemento 0,5
elemento 2
elemento 2,5
elemento 3
```

## Operação: Remover de uma posição específica



Importante: `validar índice`.

Casos especiais: `remover início`, `remover do final`



```
169 public Node remove(int index){ 1 usage
170     if (index < 0 || index >= length) return null;
171     if (index == 0) return removeFirst();
172     if (index == length - 1) return removeLast();
173
174     Node prev = get(index - 1);
175     Node temp = prev.next;
176
177     prev.next = temp.next;
178     temp.next = null;
179     length--;
180
181     return temp;
182 }
183
184 public static void main(String[] args) {
185
186     LinkedList list = new LinkedList( data: "elemento 1");
187
188     list.append("elemento 2");
189     list.append("elemento 3");
190     list.prepend( data: "elemento 0");
191     list.remove( index: 2);
192     list.print();
193 }
194 }
195
```

Implementando o método

Imprimindo o método

Run LinkedList x

```
"E:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.4\lib\idea_rt.jar=54368:C:\Program Files\Jet
elemento 0
elemento 1
elemento 3

Process finished with exit code 0
|
```

## Operação: Remover de uma posição específica



Importante: `validar índice`.

Casos especiais: `remover início`, `remover do final`

Hora de codificar!

## Vantagens:

- 1 - Estrutura dinâmica;
- 2 - Utilização da memória;
- 3 - Utilizado na construção de outras estruturas;

## Desvantagens:

- 1 - Acesso sequencial;
- 2 - Complexidade de implementação;