

Google



olabi

# ◀ESTRUTURADE DADOS▶



## { Estrutura de dados

Com a estrutura de dados, encontramos maneiras mais eficientes de acessá-los, uma vez que, ao lidar com estas estruturas, não nos concentramos apenas em um dado, mas em diferentes conjuntos de dados e em como eles se relacionam uns com os outros de maneira organizada.

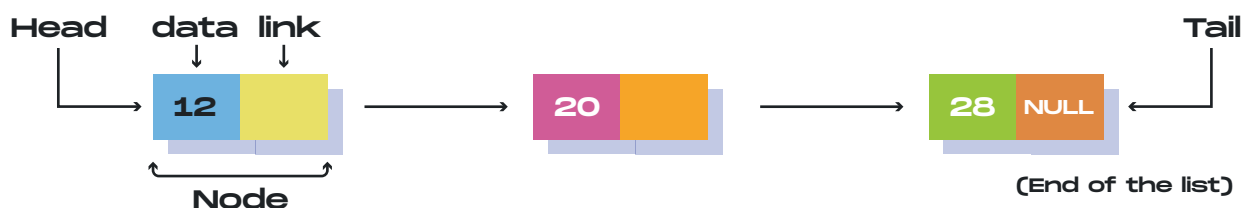
Na apostila anterior, estudamos sobre *array* (ou matriz, em português): uma estrutura de dados linear baseada em índice, o que significa que cada elemento é acessado por meio deste. Estes dados são armazenados na memória de forma sequencial, ocupando posições contíguas de memória. O fato da *array* ser uma estrutura de tamanho fixo impede seu crescimento de forma dinâmica, então, sempre que for preciso aumentar a capacidade da *array*, será preciso criar uma nova e transferir todos os elementos da original para essa nova instância. Esse processo exige a criação de uma nova *array*, que suporte um tamanho maior.

Nos tópicos de hoje, falaremos sobre estruturas de dados lineares dinâmicas. Nestas, ao contrário de estruturas baseadas em *arrays*, objetos são criados e removidos sob demanda.

## { LinkedList

LinkedList (lista vinculada, em inglês) é uma estrutura de dados lineares dinâmicos na qual elementos não são armazenados em locais de memória contíguos. Eles apresentam uma coleção de objetos encadeados chamados de *nodes* (ou nós, em português), que carregam referências para seus vizinhos e são armazenados aleatoriamente na memória.

Os *nodes* guardam a informação que queremos manipular e as referências de mesmo tipo para seus vizinhos. Eles apresentam dois campos: dados armazenados naquele endereço em particular e o ponteiro, que contém o endereço do próximo *node* na memória. O último *node* da lista contém o ponteiro para o nulo.



## { ArrayList vs. LinkedList

A classe LinkedList pode conter muitos objetos do mesmo tipo, como uma ArrayList. Ambas implementam a interface da lista que permite adicionar, alterar, ou remover itens da lista, além de limpá-la. Entretanto, embora as classes ArrayList e LinkedList possam ser usadas da mesma forma, elas são construídas de maneiras diferentes.

### **ArrayList**

A classe ArrayList tem uma matriz regular dentro dela. Quando um elemento é adicionado, ele é colocado dentro da matriz. Se a *array* não for suficientemente grande, uma nova e maior é criada para substituir a antiga, que é removida.

### **LinkedList**

A LinkedList armazena seus itens em ambientes tipo container. A lista tem um link para o primeiro recipiente; cada recipiente, um link para o próximo da lista. Para adicionar um elemento, ele é colocado em um novo recipiente e este é ligado a um dos outros da lista.

Use uma **ArrayList** para armazenar e acessar dados, e **LinkedList** para manipular os dados



## { Métodos de LinkedList

A LinkedList fornece vários métodos para realizar determinadas operações de forma mais eficiente:

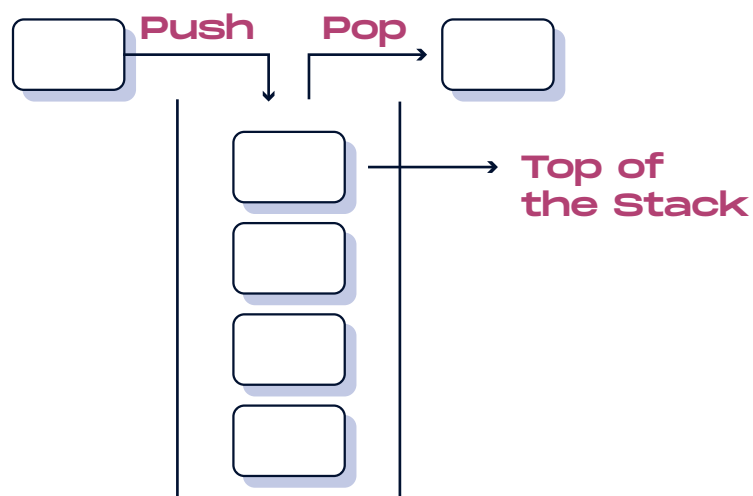
<b>addFirst()</b>	Adiciona um item ao início da lista
<b>addLast()</b>	Adiciona um item ao final da lista
<b>removeFirst()</b>	Remove um item do início da lista
<b>removeLast()</b>	Remove um item do final da lista
<b>getFirst()</b>	Obtém o item do início da lista
<b>getLast()</b>	Obtém o item do final da lista

## { Stack

*Stack* (ou pilha, em português) é uma estrutura de dados linear usada para armazenar a coleção de objetos. Ela é baseada na premissa *Last-In-First-Out* (LIFO, em inglês): o último elemento armazenado é obrigatoriamente o primeiro a ser retirado.

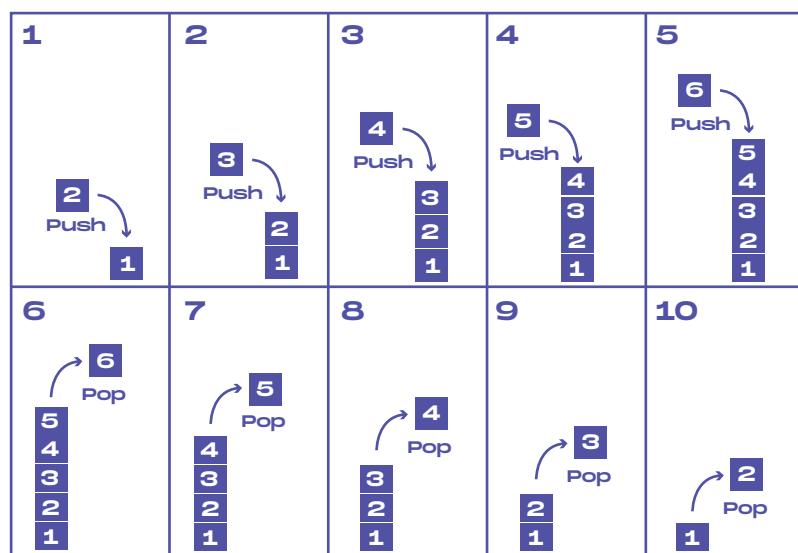
Uma maneira fácil de explicar como essa estrutura de dados funciona é compará-la a uma pilha de pratos, na qual o primeiro prato a entrar na pilha será o último a sair (em português, “*Last-In-First-Out*” significa “último a entrar, primeiro a sair”).

Os dois métodos fundamentais associados a uma estrutura de pilha devem permitir adicionar um elemento ao topo da pilha (**push()**) ou recuperar/ remover o elemento no topo da pilha (**pop()**).



Outras funcionalidades associadas a esta classe incluem os métodos:

- **peek()** - qual o objeto que está no topo da pilha?
- **search()** - em qual profundidade se encontra o objeto especificado?
- **empty()** - a pilha está vazia?



## { Empty Stack

*Empty Stack* (pilha vazia, em português) ocorre quando a pilha não apresenta nenhum elemento. Nesse caso, o valor da variável superior é -1.

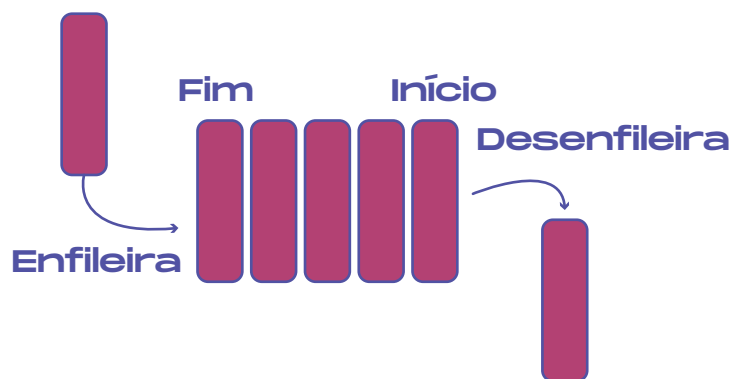
## { Diferentes valores de topo (top value, em inglês)

Top value	Significado
-1	A pilha está vazia
0	A pilha tem apenas um elemento
N-1	A pilha está cheia
N	A pilha está transbordando

## { Queue

*Queue* (fila, em português) é uma estrutura de dados que ordena o elemento FIFO (*First-In-First-Out*). No FIFO, o primeiro item da fila será o primeiro a ser consumido; ao ser consumido, é removido do grupo de itens passando o segundo item para primeiro, o terceiro para segundo e assim por diante, até o final da fila. Uma analogia para explicar esse algoritmo seria uma fila de supermercado, na qual pessoas são atendidas por ordem de chegada (*"First-In-First-Out"*, ou "primeiro a chegar, primeiro a sair" em português).

Filas normalmente tem apenas dois métodos: um para adição de novos itens chamado de *enqueue* (enfileirado, em português), e outro para remoção de um item chamado de *dequeue* (desenfileirado, em português).



## { Empty Stack

Método	Descrição
<b>add()</b>	Insere um elemento na fila e retorna uma exceção quando não há mais adições possíveis
<b>offer()</b>	Insere um elemento na fila e retorna true (verdadeiro) ou false (falso), caso a adição seja feita
<b>remove()</b>	recupera e remove o primeiro elemento da fila
<b>poll()</b>	retorna e remove o primeiro elemento da fila, ou retorna nulo se ela estiver vazia
<b>element()</b>	retorna o primeiro elemento da fila sem removê-lo, ou como exceção se ela estiver vazia
<b>peek()</b>	retorna o primeiro elemento da fila sem removê-lo, ou nulo se ela estiver vazia

### Referências

#### LinkedList

<https://www.javatpoint.com/singly-linked-list>

[https://www.w3schools.com/java/java\\_linkedlist.asp](https://www.w3schools.com/java/java_linkedlist.asp)

<https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html>

<https://www.geeksforgeeks.org/data-structures/linked-list/>

#### Stack

<https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html>

<https://www.javatpoint.com/java-stack>

<https://www.geeksforgeeks.org/stack-class-in-java/>

#### Queue

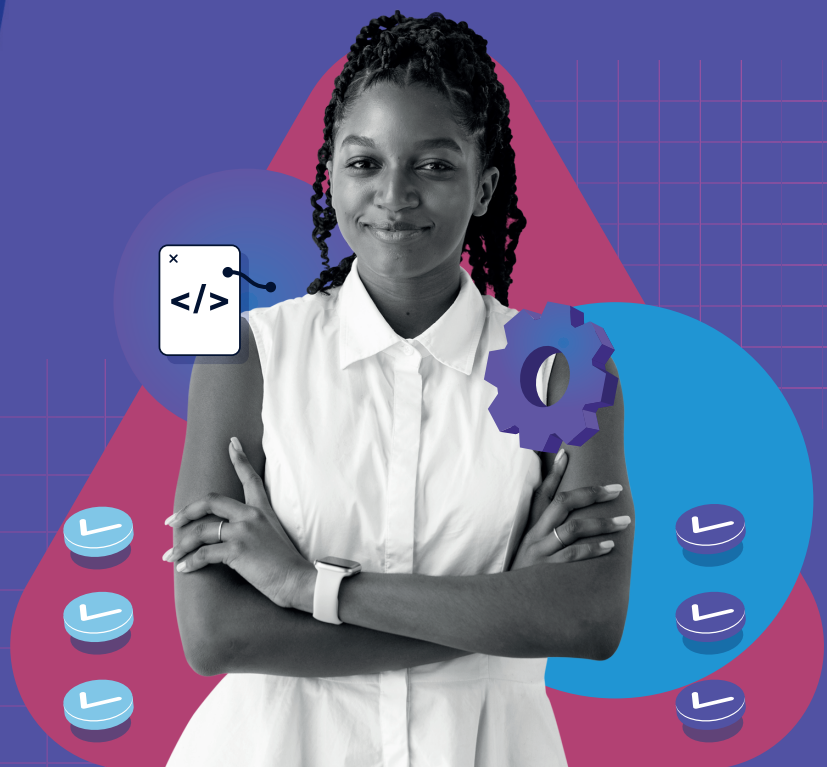
<https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html>

<https://www.javatpoint.com/java-priorityqueue>

<https://www.geeksforgeeks.org/queue-interface-java/>

# ◀ESTRUTURA DE DADOS▶

## /exercícios/





## { Exercício coletivo I

Escreva um programa para unir duas listas vinculadas.

## { Exercício coletivo II:

Escreva um programa para remover e retornar o primeiro elemento de uma lista vinculada.

## { Exercício coletivo III:

Implemente uma fila usando pilhas, ou seja: enfileire adicionando um elemento ao final da fila e desenfileire removendo um elemento do início da fila, de forma a utilizar as funções padrão da estrutura de dados da pilha.

## { Exercício coletivo IV:

Escreva um programa para inverter uma fila.

## { Exercício coletivo V:

Crie um algoritmo para reverter os primeiros K elementos de uma fila.

## { Extras

Escreva um programa para iterar por meio dos elementos de uma lista vinculada e começar na posição especificada.

Escreva um programa de troca de dois elementos de uma lista vinculada.

Escreva um programa para comparar duas filas prioritárias (Dica: PriorityQueue)

Escreva um programa para mudar a fila de prioridade *Queue* para fila de prioridade máxima (Dica: PriorityQueue).

### <Leitura recomendada>

<https://www.geeksforgeeks.org/collections-in-java-2/?ref=lbp>

### <Referência dos exercícios>

<https://www.w3resource.com/java-exercises/collection/>

