



# Módulo 11





# BackEnd Java

---

Jeferson Tigik



A decorative graphic on the left side of the slide. It features a large cyan hexagon with the number '3' inside. Surrounding this central hexagon are several smaller hexagons and icons: a lightbulb, a thumbs up, a network node, a smartphone, a magnifying glass, a gear, and a speech bubble. The background is a dark blue gradient.

3

Queue

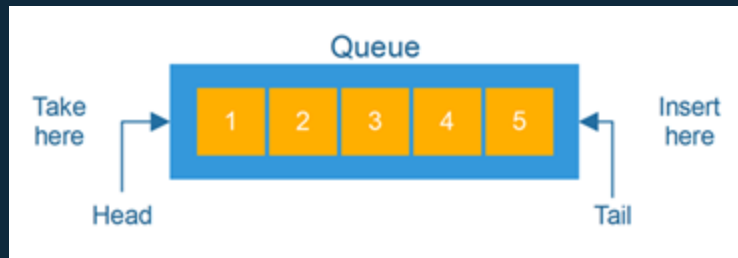
# Implementações Deque/Queue

- **ArrayDeque:** Esta é uma classe especial que implementa uma estrutura de dados de fila de duas extremidades, onde pode inserir e remover elementos de ambas as extremidades. Suporta a implementação de um array redimensionável que cresce automaticamente.



# Implementações - Queue

- **PriorityQueue:** Nesta classe, o elemento é inserido na parte de trás da fila. Esta operação é chamada de enfileiramento. Este mesmo elemento sai a partir da frente da fila, operação chamada de desenfileiramento. Esse procedimento de entrada e saída recebe o nome de fila, ou FIFO (first-in first-out), ou seja, “primeiro a entrar, primeiro a sair”. Seus métodos principais são:



A decorative graphic on the left side of the slide. It features a large central hexagon with a blue-to-teal gradient, containing the number '4'. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and teal. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines. The entire graphic is set against a dark blue background.

4

Set



# Implementações - Set

→ HashSet: o acesso aos dados é mais rápido que em um TreeSet, mas nada garante que os dados estejam ordenados. Escolha este conjunto quando a solução exigir elementos únicos e a ordem não for importante.

Exemplo de aplicação: Poderíamos usar esta implementação para criar um catálogo pessoal das canções da nossa discografia;





# Implementações - Set

- **TreeSet**: os dados são classificados, mas o acesso é mais lento que em um HashSet. Se a necessidade for um conjunto com elementos não duplicados e acesso em ordem natural, prefira o TreeSet. É recomendado utilizar esta coleção para as mesmas aplicações de HashSet, com a vantagem dos objetos estarem em ordem natural;







# Implementações - Set

- LinkedHashSet: é derivada de HashSet, mas mantém uma lista duplamente ligada através de seus itens. Seus elementos são iterados na ordem em que foram inseridos. Opcionalmente é possível criar um LinkedHashSet que seja percorrido na ordem em que os elementos foram acessados na última iteração.

Exemplo de aplicação: Poderíamos usar esta implementação para registrar a chegada dos corredores de uma maratona;





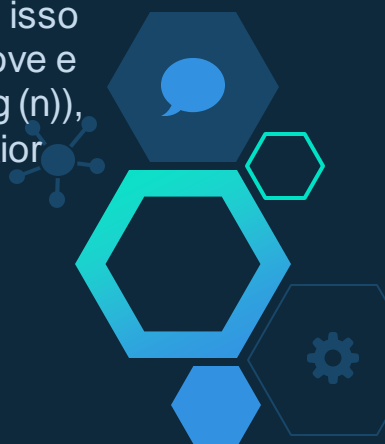
# Diferenças entre - Set

- **HashSet:** O HashSet é o mais rápido de todos e seus elementos não são ordenados e não importa o quanto você adicione, remova, retire, o tempo de execução sempre será o mesmo. Por outro lado, a garantia de continuidade na ordem dos elementos inseridos é zero, ou seja, esse tipo de estrutura é indicada se você precisa apenas garantir a alta performance sem se importar com a ordem com que os elementos estão ordenados.





# Diferenças entre - Set

- **TreeSet:** Sua principal característica é que ele é o único Set que implementa a interface SortedSet em vez de Set diretamente, mas de qualquer forma SortedSet implementa Set, assim continuamos tendo os mesmos métodos no TreeSet. Pelo fato de ele implementar SortedSet ele possui elementos ordenados automaticamente, ou seja, independente da ordem que você inserir os elementos, eles serão ordenados. Mas isso tem um custo, a complexidade para os métodos add, remove e contains são bem maiores que do HashSet, são elas  $O(\log(n))$ , não é bem uma complexidade exponencial mas é bem maior que  $O(1)$  que tem seu tempo inalterado.
- 



# Diferenças entre - Set

- **LinkedHashSet:** É um meio termo entre HashSet e TreeSet, ou seja, ela nos proporciona um pouco da performance do HashSet e um pouco do poder de ordenação do TreeSet. O LinkedHashSet faz uso também do HashTable com linked list, ou seja, temos aqui a seguinte situação: Os elementos continuam na ordem que são inseridos, diferente do HashSet que “embaralha” tudo.

