



Módulo 11





BackEnd Java

Rodrigo Pires



A decorative graphic on the left side of the slide. It features a large, central cyan-to-blue gradient hexagon. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, a gear, and a speech bubble. There is also a small network-like icon with a central node and radiating lines.


Coleções - Collections

Parte 1



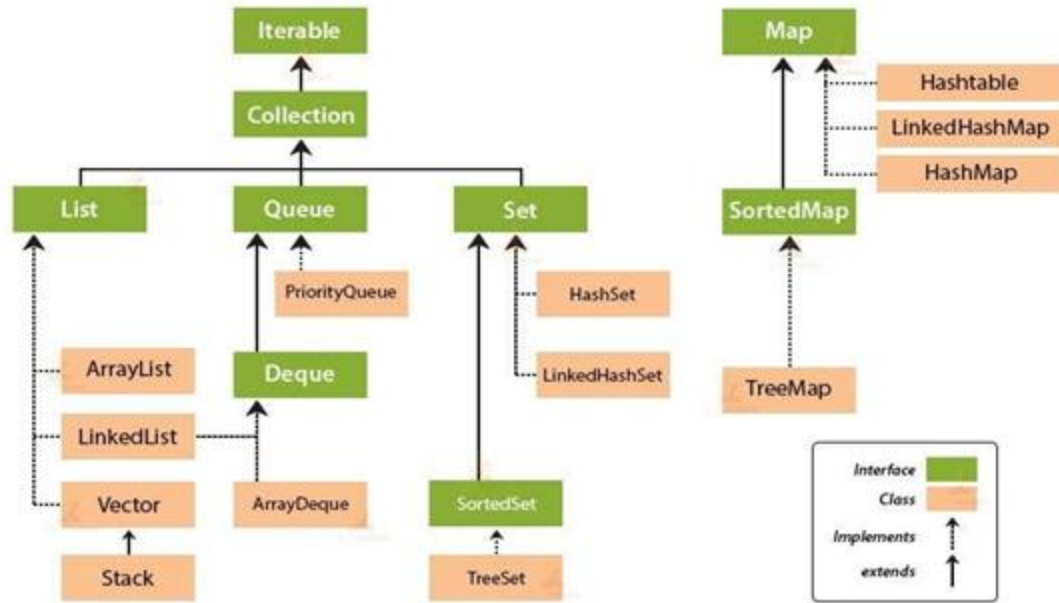
Coleções - Collections

É um conjunto bem definido de interfaces e classes para representar e tratar grupos de dados como uma única unidade, que pode ser chamada coleção, ou collection. A Collections contém os seguintes elementos:

- **Interfaces:** tipos abstratos que representam as coleções. Permitem que coleções sejam manipuladas tendo como base o conceito “Programar para interfaces e não para implementações”, desde que o acesso aos objetos se restrinja apenas ao uso de métodos definidos nas interfaces.
 - **Implementações:** são as implementações concretas das interfaces.
 - **Algoritmos:** são os métodos que realizam as operações sobre os objetos das coleções, tais como busca e ordenação.
- 

Coleções - Collections

Collection Framework Hierarchy in Java






Interfaces

- **Iterator**: está no topo da hierarquia e possibilita percorrer uma coleção e remover seus elementos;
- **Collection**: Não existe implementação direta dessa interface, mas ela define as operações básicas para as coleções, como adicionar, remover, esvaziar, etc.;
- **List**: define uma coleção ordenada, podendo conter elementos duplicados. Em geral, o usuário tem controle total sobre a posição onde cada elemento é inserido e pode recuperá-los através de seus índices. Prefira esta interface quando precisar de acesso aleatório, através do índice do elemento;





Interfaces

- **Queue**: um tipo de coleção para manter uma lista de prioridades, onde a ordem dos seus elementos, definida pela implementação de Comparable ou Comparator, determina essa prioridade. Com a interface fila pode-se criar filas e pilhas;
 - **Map**: mapeia chaves para valores. Cada elemento tem na verdade dois objetos: uma chave e um valor. Valores podem ser duplicados, mas chaves não. SortedMap é uma interface que estende Map, e permite classificação ascendente das chaves. Uma aplicação dessa interface é a classe Properties, que é usada para persistir propriedades/configurações de um sistema, por exemplo.
 - **Set**: interface que define uma coleção que não permite elementos duplicados. A interface SortedSet, que estende Set, possibilita a classificação natural dos elementos, tal como a ordem alfabética;
- 



1 e 2

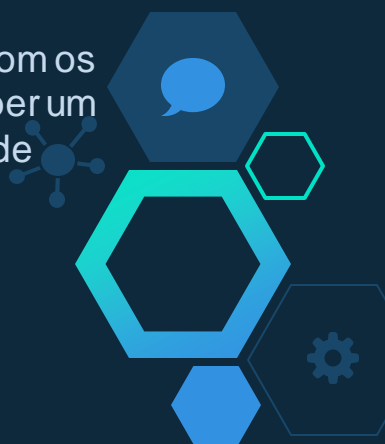
List



Implementações - List

→ ArrayList: É como um array cujo tamanho pode crescer. A busca de um elemento é rápida, mas inserções e exclusões não são. Podemos afirmar que as inserções e exclusões são lineares, o tempo cresce com o aumento do tamanho da estrutura. Esta implementação é preferível quando se deseja acesso mais rápido aos elementos.

Exemplo de aplicação: Se você quiser criar um catálogo com os livros de sua biblioteca pessoal e cada obra inserida receber um número sequencial (que será usado para acesso) a partir de zero;





Implementações - List

→ LinkedList: Implementa uma lista ligada, ou seja, cada nó contém o dado e uma referência para o próximo nó. Ao contrário do ArrayList, a busca é linear e inserções e exclusões são rápidas. Portanto, prefira LinkedList quando a aplicação exigir grande quantidade de inserções e exclusões.

Exemplo de aplicação: Um pequeno sistema para gerenciar suas compras mensais de supermercado pode ser uma boa aplicação, dada a necessidade de constantes inclusões e exclusões de produtos;





Diferenças entre - List

- **ArrayList:** É mais rápido quando utilizado para consultar, já para inserção e exclusão, são mais lentas.
- **LinkedList:** É mais rápido quando utilizado para inclusão e exclusão de dados com um grande volume de dados. Também pode ser utilizado como fila(Queue).



A decorative pattern of hexagons in various shades of blue and cyan on the left side of the slide. Some hexagons contain icons: a lightbulb, a thumbs up, a network node, a smartphone, a magnifying glass, a gear, and a speech bubble.

3

Queue

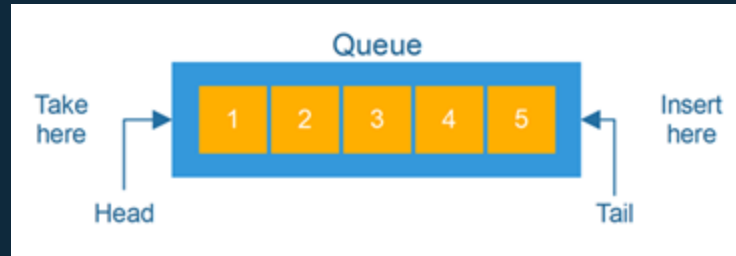
Implementações - Deque/Queue

- **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, and a speech bubble icon. 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;





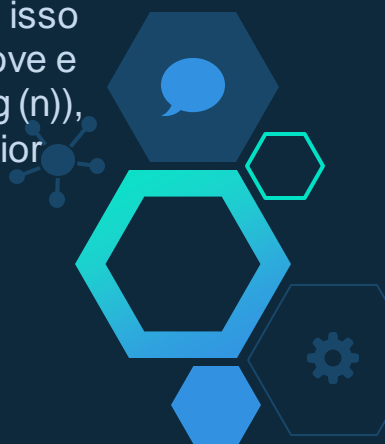
Diferenças entre - Set

- **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.





Tarefa

- Fazer um programa que receba uma lista de nomes e ordene em ordem alfabética.
- Versionar no github na pata Colecoes-parte-1





Referências

[Exemplos disponíveis no meu github:](#)

<https://github.com/digaomilleniun/backend-java-ebac>

