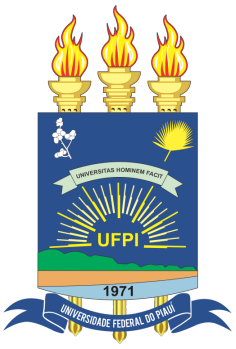

Universidade Federal do Piauí
Alunos: Jhoisnáyra Vitória Rodrigues de Almeida
Pedro Gonçalves Neto.
Curso: Ciência da Computação.

Relatório de Estrutura de Dados: *Java Collections API*.

Teresina, 24 de novembro de 2019.

.

1. Java Collections

O framework Java Collections é um conjunto de classes e interface que implementam estruturas de dados comumente utilizadas, como pode ser observado nas imagens 1 e 2. Embora seja visto como uma estrutura, o Java Collections funciona como uma biblioteca, fornecendo interfaces que são implementadas de forma diferente por diversas classes.

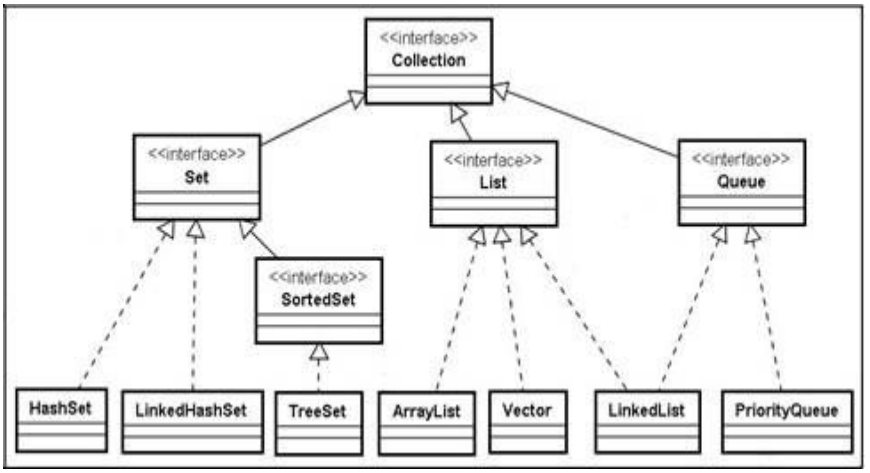

Imagem 1 - Interfaces e classes que derivam de Collection

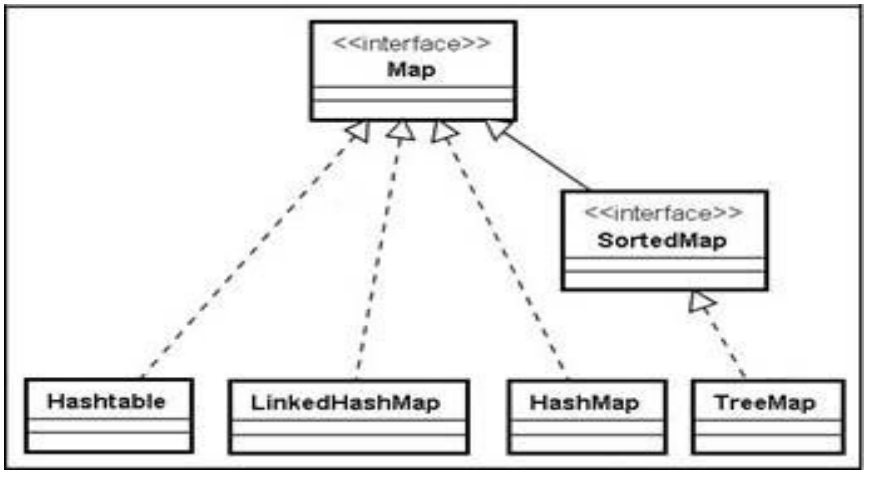

Imagem 2 - Interface e classes que derivam de Map

2. Collection

Está no topo da hierarquia das interfaces, porém não existe implementação direta dessa interface. Ela define as operações básicas para as coleções, como adicionar, remover, esvaziar.

3. List

A interface 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-lo através de seus índices. Esta interface é preferível quando precisa-se de um acesso aleatório, através do índice do elemento.

3.1. Vector

Vector é a mais antiga das implementações, já existia na versão 1.0 do Java, antes mesmo da inclusão da API Collection. Vector e ArrayList são muito parecidos, porém, há uma grande diferença entre as duas implementações: O Vector é sincronizado, e o ArrayList não. Quer dizer que caso sua aplicação necessite ser thread-safe em determinado ponto, o uso de Vector garantirá isso, essa é a vantagem do uso desta Classe. Pelo mesmo fato de ser sincronizado, tem-se sua desvantagem, pois isto torna as operações em cima desta coleção muito mais custosas e consequentemente mais lentas. Um ponto importante também é que o Vector aumenta seu tamanho em 100% quando a estrutura está cheia.

3.2. LinkedList

LinkedList é a implementação para uma lista ligada. Fornece métodos adicionais para tratar as extremidades da lista, o que torna possível a obtenção e a remoção do primeiro ou último elemento da lista. Seu funcionamento interno é diferente do ArrayList e possui melhor performance nos métodos *add* e *remove*, em compensação, seus métodos *get* e *set* possuem uma performance pior.

3.3. ArrayList

Arraylist é a implementação mais utilizada, ela trabalha com um array interno para gerar uma lista. Portanto, ela é mais rápida na pesquisa do que a LinkedList. Mesmo contendo a palavra *array*, ArrayList não é um array, ela usa um array como estrutura para armazenar dados, porém este atributo está encapsulado e o mesmo é inacessível. Um ponto importante é que o ArrayList aumenta seu tamanho em 50% quando a lista está cheia. Pode ser considerada uma desvantagem da estrutura o fato de possuir apenas uma dimensão, diferente de um array, que pode ter mais de uma.

4. Set

A interface Set define uma coleção que não permite elementos duplicados. SortedSet é uma interface que estende Set e possibilita ainda a classificação natural dos elementos, tal como a ordem alfabética.

4.1. HashSet

HashSet é a implementação que garante melhor performance comparada às outras relacionadas a Set. Esta usa HashTable e seus elementos não são ordenados, a complexidade desta estrutura é O(1), em outras palavras, não importa o quanto é adicionado ou removido, o tempo de execução será sempre o mesmo. Por outro lado, não há garantia quanto à continuidade da ordem dos elementos inseridos, ou seja, esse tipo de estrutura é indicada quando a ordenação não importa, e sim a performance.

4.2. LinkedHashSet

LinkedHashSet é um meio termo entre HashSet e TreeSet, ou seja, proporciona um pouco da performance do HashSet e um pouco da ordenação do TreeSet. O LinkedHashSet faz uso também do HashTable com lista encadeada, ou seja, os elementos continuam na ordem em que foram inseridos, diferente do HashSet. A complexidade do LinkedHashSet é O(1) para operações básicas.

4.3. TreeSet

TreeSet implementa uma árvore rubro-negra. Sua principal característica é que ele é o único Set que implementa a interface SortedSet em vez de Set diretamente. Por esse fato, ele possui elementos ordenados automaticamente, ou seja, independente da ordem de inserção. Entretanto, isso tem um custo, a complexidade para os métodos *add*, *remove* e *contains* são bem maiores que do HashSet, são elas O(log(n)). Por implementar SortedSet, o TreeSet também oferece alguns métodos adicionais, como por exemplo: *first(), last(), headSet(), tailSet()*.

5. Map

A interface Map mapeia chaves para valores. Cada elemento tem na verdade dois objetos: uma chave e um valor. Valores podem ser duplicados, mas chaves têm de ser únicas. SortedMap é uma interface que estende Map, e possibilita a classificação ascendente das chaves. Uma aplicação dessa interface é a classe Properties, que é usada para persistir propriedades/configurações de um sistema, por exemplo.

5.1. HashMap

HashMap baseia-se em uma tabela de espalhamento, ou Hash Table. Pode ser vantajosa a utilização do HashMap onde seja necessário um identificador, porém pode ser um ponto negativo caso a ordenação importe.

5.2. LinkedHashMap

LinkedHashMap mantém uma lista duplamente encadeada através dos seus itens. A ordem de iteração é a ordem em que as chaves são inseridas no mapa.

5.3. TreeMap

TreeMap é a implementação da interface SortedMap, onde há uma garantia de que o mapa estará classificado em ordem ascendente das chaves. Entretanto, existe a opção de especificar uma ordem diferente, sobrescrevendo o método *compareTo()*. Apesar de ser bom para ordenação de dados, o TreeMap perde no quesito desempenho para o HashMap, por operações básicas serem mais custosas.

6. Implementação de Java Collections

No desenvolvimento do trabalho, decidiu-se criar uma classe para executar os 9 tipos de collections, criando métodos estáticos que efetuam a inserção, busca e exclusão das

palavras listadas (Lisbon, NASA, Kyunghee, Konkuk, Sogang, momentarily, rubella, vaccinations, government, Authorities). Dentro de cada método, usa-se a classe Stopwatch, para calcular o tempo de execução de cada aplicação, o resultado desse cálculo é inserido em uma lista de tempos e transferido por parâmetro para a classe que cria o gráfico de desempenho.

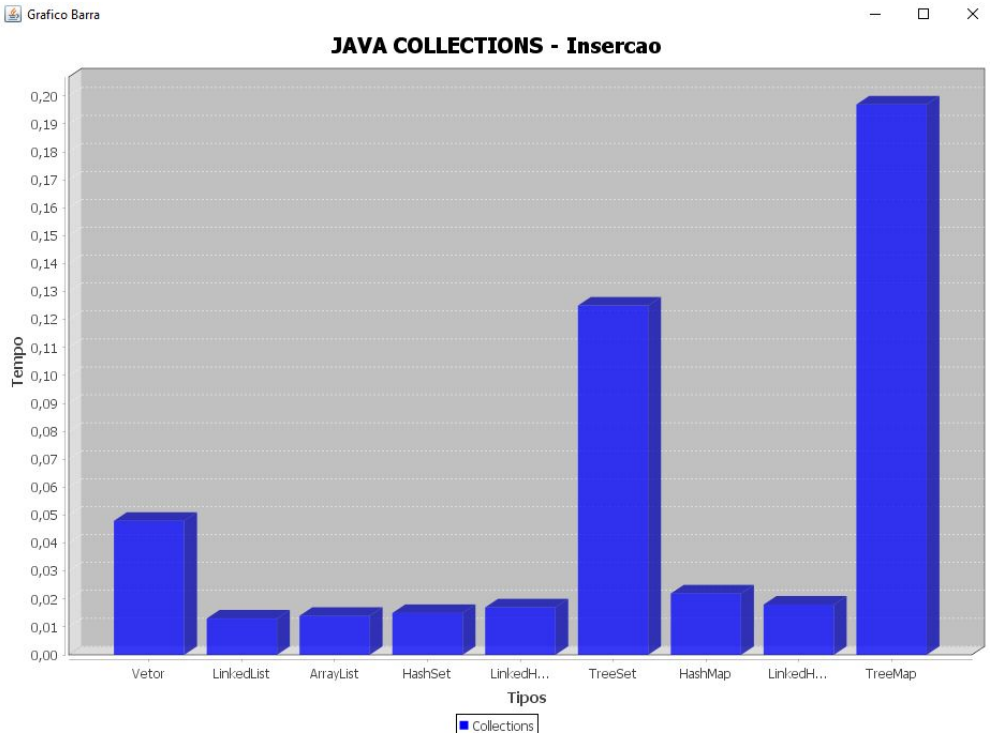

Imagem 3 - Gráfico de desempenho da inserção.

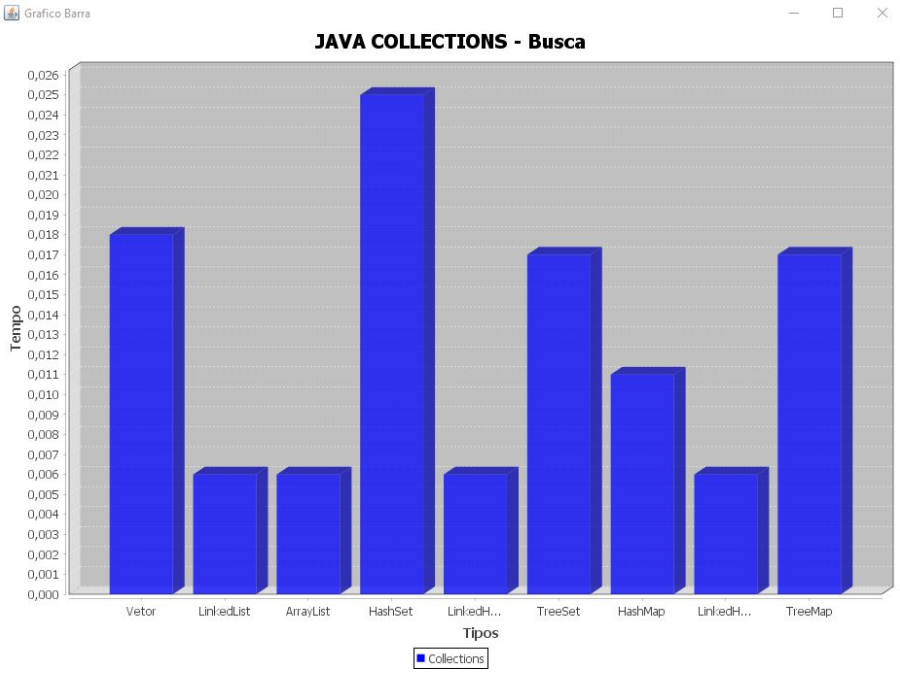

Imagem 4 - Gráfico de desempenho da busca de palavras.

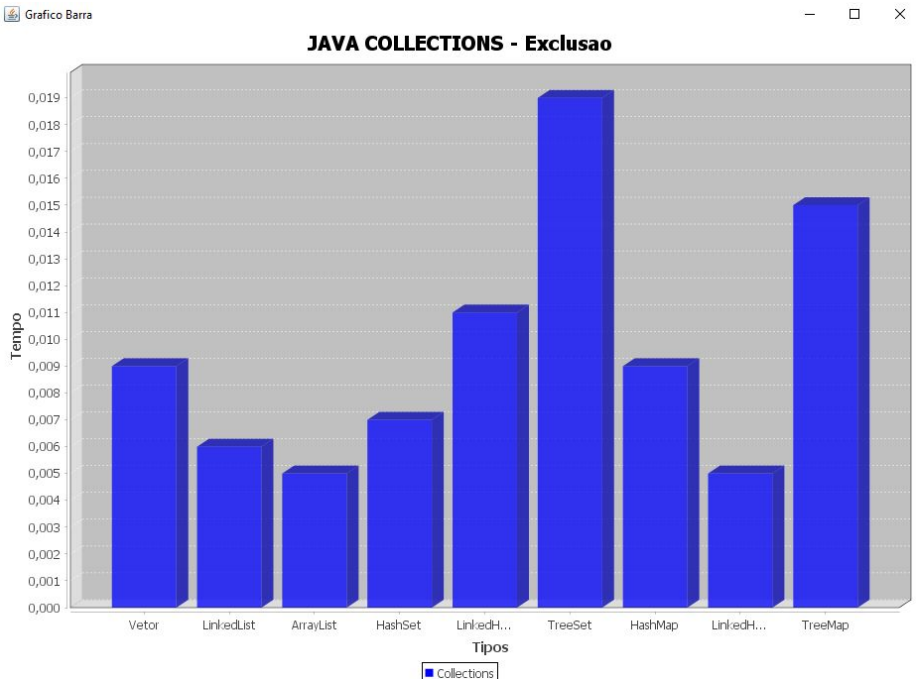

Imagem 5 - Gráfico de desempenho da exclusão de palavras.

7. Priority Queue

7.1. Descrição

PriorityQueue é uma classe que implementa a interface Queue, que por sua vez, estende da interface Collection. Nessa estrutura, os dados são armazenados em sua ordem natural, o que pode ser considerado uma vantagem num sistema de senhas, por exemplo, pois será feito o atendimento na ordem natural dos números inteiros, caso este seja usado como tipo de dado da fila prioritária. Essa característica pode ainda ser alterada, implementando uma fila com prioridade que é parametrizada por um objeto que implementa a interface *comparable.* Por outro lado, uma das desvantagens da PriorityQueue é que não é possível alterar a prioridade de um elemento depois de inserido.

7.2. Implementação

Na implementação da PriorityQueue, foram criadas duas filas, uma para a fila normal e outra para a fila com prioridade. Ambas foram parametrizadas com o tipo de dado *Clientes*, que foi a classe definida para implementar a interface *comparable*, sendo assim possível sobrescrever o método *compareTo()*. A prioridade foi dada de acordo com o índice de inserção dos elementos na fila, seguiu-se a política FIFO (First-in, First-out) em cada uma das filas. Essa política só é ignorada se a as duas filas forem vistas como uma só, neste caso, se

houverem elementos na fila prioritária, estes elementos serão removidos antes mesmo de qualquer elemento inserido antes deles na fila normal.

8. Referências:

ALGS4. **Java Algorithms and Clients**. Disponível em: https://algs4.cs.princeton.edu/code/ . Acesso em: 25 nov. 2019.

EDERMFL. **COLLECTIONS #1: ARRAYLIST, LINKEDLIST E VECTOR**. Disponível em: https://edermfl.wordpress.com/2016/03/23/collections-1-arrayslist-linkedlist-e-vector/. Acesso em: 24 nov. 2019.

DEVMEDIA. **Diferenças entre TreeSet, HashSet e LinkedHashSet em Java**. Disponível em: https://www.devmedia.com.br/diferencas-entre-treeset-hashset-e-linkedhashset-em-java/29077. Acesso em: 24 nov. 2019.

DEVMEDIA. **Java Collections: Como utilizar Collections**. Disponível em: https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450. Acesso em: 24 nov. 2019.

ORACLE. **Class PriorityQueue**. Disponível em: https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html. Acesso em: 25 nov. 2019.