**Collections – DIO**

**Coleções com iterações ordenadas**

**Java.util.List**

* Permite elementos duplicados
* Garante ordem de inserção

extends  
implements

<< interface>>  
Collection

<< interface>>  
List

<< interface>>  
Queue

PriorityQueue

LinkedList

Vector

ArrayList

**Vector**

* Implementação Thread safe

(***Classes Thread safe*** - Implementam algum mecanismo para evitar que múltiplas threads alterem alguma informação do objeto de forma concorrente)

**ArrayList X LinkedList**

* ***ArrayList*** deve ser usada onde mais operações de pesquisa/busca são necessárias
* ***LinkedList*** deve ser usada onde mais operações de inserção e exclusão são necessárias.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| Usa um array dinâmico por debaixo dos panos. Usa *Generics* para manipular qualquer tipo de objeto. | Essa classe usa uma ***lista duplamente ligada***.Também usa *Generics* para manipular qualquer tipo de objeto. |
| Manipulação na ArrayList leva mais tempo devido a implementação interna. Sempre que removemos um elemento, o array é cruzado (*traversed*) e os bits são deslocados (*shifted*). | Manipulaçao na LinkedList é mais rápida que na ArrayList, pois em listas duplamente ligadas, não há deslocamento (*shift*) de bits na memória. A lista é cruzada (*traversed*) e as referências são modificadas. |
| Implementa a ***interface List*** | Implementa as ***interfaces List e Queue*** |
| Funciona melhor quando a aplicação precisa **salvar e acessar** os dados. | Funciona melhor quando necessário **manipular** os dados (**insert, delete**) |

**Coleções Singulares – Set**

**Java.util.Set**

* Não permite elementos duplicados
* Não possui índice

<< interface>>  
Collection

extends  
implements

<< interface>>  
Set

<< interface>>  
SortedSet

HashSet

LinkedHashSet

<< interface>>  
NavigableSet

TreeSet

\* TreeSet é usado quando há necessidade de ordenação

|  |  |  |
| --- | --- | --- |
| **HashSet** | **LinkedHashSet** | **TreeSet** |
| Usa internamente um ***HashMap*** para salvar seus elementos. | Usa internamente um ***LinkedHashMap*** para salvar seus elementos. | Usa internamente um ***TreeMap*** para salvar seus elementos. |
| **Não mantem ordenação** de elementos. | Mantém os elementos na **ordem de inserção**. | Por padrão, mantém a **ordenação natural dos elementos**. Ex: ordem numérica, alfabética, etc |
| Oferece **melhor performance** que LinkedHashSet e TreeSet. | Possui **performance mediana**: melhor que TreeSet e pior que HashSet. | **Pior performance** entre os 3: LinkedHashSet, HashSet e TreeSet. |
| Permite **no máximo um elemento nulo**. | Também permite **no máximo um elemento nulo**. | Não permite **nenhum elemento nulo**. |

**Coleções de pares – Map**

**Java.util.Map**

* Elementos únicos (***key***) para cada valor (***value***)
* As **chaves são únicas**, não se repetem
* Não etende a interface Collection

<< interface>>  
Map

<< interface>>  
SortedMap

Hashtable

Hashmap

LinkedHashMap

<< interface>>  
NavigableMap

TreeMap

extends  
implements

\* TreeSet é usado quando há necessidade de comparação e ordenação

|  |  |  |  |
| --- | --- | --- | --- |
| **Propriedade** | **HashMap** | **LinkedHashMap** | **TreeMap** |
| **Complexidade do algoritmo (Big O Notation).**  **Get, Put, Contains Key e Remove** | O(1) | O(1) | O(log n)  (mais demorado) |
| **Ordem de Iteração** | Randômica | Ordenado de acordo com a **Ordem de Inserção** ou **Ordem de Acesso** (especificada durante a construção) | Ordenado de acordo com **a Ordem Natural das Keys** ou **Comparator** (especificada durante a construção) |
| **Keys Nulas** | Permitidas | Permitidas | Não permitida se a Key usar **Ordenação Natural** ou **Comparator** não permitir comparação com keys nulas |
| **Interface Implementada** | Map | Map | Map, SortedMap e NavigableMap |
| **Sincronização** | Nenhuma, usa ***Collections. synchronizedMap()*** | Nenhuma, usa ***Collections. synchronizedMap()*** | Nenhuma, usa ***Collections. synchronizedMap()*** |
| **Estrutura de Dados** | ***List of Buckets***, se tiver mais de 8 entradas no bucket, Java 8 irá mudar para uma Tree balanceada de Linked List | **Lista duplamente encadeada** de Buckets | **Red-Black Tree**  (um tipo de árvore de busca binária propriamente balanceada) Implementação de Binary Tree. Essa ED oferece O(log n) para insert, delete e search e O(n) como complexidade de espaço |
| **Aplicações** | **Propósito geral**, **rápido retorno**, **não-sincronizado**. *ConcurrentHashMap* pode ser usado em caso de concorrência | Pode ser usado para algoritmo **LRU cache** (menos usado recentemente), ou outros casos em que a **ordem de inserção ou acesso** importem | Algoritmos em que features ***Sorted*** ou ***Navigable*** são necessárias. Por ex.: Encontrar dentro de uma lista de empregados, quais tem salário próximo |
| **Requisitos das Keys** | **Equals()** e **hashcode()** precisam ser sobrescritos | **Equals()** e **hashcode()** precisam ser sobrescritos | ***Comparator*** precisa ser fornecido para a implementação da Key, caso contrário a **Ordem Natural** será usada para ordenação |

**Java Streams API**

**Classes Anônimas**

É uma classe que **não recebeu um nome** e é tanto **declarada** como **instanciada** em **uma única instrução**.  
Você deve **considerar** o uso de classe anônima sempre que precisar criar uma **classe que será instanciada somente uma vez**.

* A ideia é que a classe seja **declarada no mesmo lugar em que será usada**.
* O uso de classes anônimas evita a criação de classes que serão instanciadas somente uma vez.

A screenshot of a computer

Description automatically generated with medium confidence

**Functional Interface**

Qualquer interface com um ***SAM* (*Single Abstract Method*)** é uma interface funcional e sua implementação pode ser tratada como expressões *lambda*.

**Obs.:** Interfaces funcionais podem ter mais de um método, mas **somente um método abstrato**.

* Annotation ***@FunctionalInterface*** pode ser usada, mas não é obrigatória.

Graphical user interface, text

Description automatically generated

**Exemplos:** Comparator, Consumer, Function, Predicate

**Lambda Expressions**

Uma função *lambda* é uma função sem declaração, ou seja, **não é preciso** colocar um **nome, um tipo de retorno e o modificador de acesso**.

* A ideia é que o método seja **declarado no mesmo lugar em que será usado**.
* As funções lambda em Java tem **sintaxe** definida como: **(argumento) -> (corpo)**

**Ex.:** (pessoa) -> (pessoa.getIdade()/2)

* *Lambda expressions* **simplificam** o uso de **classes anônimas**

Graphical user interface

Description automatically generated

**Method Reference**

Recurso do Java 8 que permite fazer referência a um método ou construtor de uma classe (**de maneira funcional**) e assim indicar que ele deve ser usado num ponto específico do código, deixando-o mais simples e legível.

Para utilizá-lo, basta informar uma **classe** **ou referência** seguida do **símbolo** **“::”** e o **nome do** **método** sem os parênteses ao final.

* **Ex.:** ClasseTeste::metodoTeste

Uso mais comum – **simplificar *lambda expressions***

Graphical user interface

Description automatically generated

**Streams API**

Traz uma nova opção para a manipulação de coleções em Java seguindo os princípios da programação funcional.   
Combinada com as expressões lambda, proporciona uma forma diferente de lidar com conjuntos de elementos, oferecendo ao desenvolvedor uma maneira simples e concisa de escrever código que resulta em facilidade de manutenção e paralelização sem efeitos indesejados em tempo de execução.

A picture containing text, clipart, screenshot

Description automatically generated