Collections

Uma Collection, segundo a Oracle, “é um objeto que agrupa vários elementos em uma única unidade”. É uma Estrutura de Dados onde é realizado o processamento de um conjunto de dados com a ajuda de métodos predefinidos.

Nesse processamento é possível:

* Armazenar,
* Recuperar,
* Manipular, e
* Comunicar dados agregados.

Este framework de coleções fornecem um conjunto de interfaces, implementações e algoritmos para armazenar e manipular grupos de objetos de maneira eficiente e flexível.

Sobre o que há nas Collections, podemos dizer que:

Interfaces: são tipos de dados abstratos que representam as coleções, e permitem que as collections sejam manipuladas e orientada para a interface e não para a implementação.

Implementações: são as implementações concretas das interfaces de collection. Podemos dizer que são estruturas de dados reutilizáveis.

Algoritmo: são os métodos que realizam as operações (cálculos úteis), como serviços de busca e organização, sobre os objetos das coleções. A característica polimórfica permite que as implementações, uma vez implementada, possam ser reaproveitadas.

BENEFICIOS DO JAVA COLLECTIONS FRAMEWORK

Reduz o esforço de programação

A Collections Framework fornece estruturas de dados e algoritmos úteis, para se concentramos diretamente no programa, e não em suas mínimas funcionalidades.

Aumenta a velocidade e qualidade do programa

As várias implementações de cada interfaces podem ser facilmente ajustadas. Não sendo necessário escrever as próprias estruturas de dados, o desenvolvedor terá mais tempo para dedicar em outras coisas, como qualidade e desempenho dos programas.

Permite interoperabilidade entre APIs não relacionadas

As interfaces de collection são nativas, por onde as APIs passam as collections de um lado para outro. Mesmo sendo escritas de modos diferentes, as APIs funcionarão de maneira perfeita

Reduz o esforço para aprender e usar novas APIs

Muitas APIs costumam pegar collections na entrada e as fornecem como saída. A proposta é reduzir ou acabar com problemas, causados a implementação de uma API e uma sub-API.

Reduz o esforço para projetar novas APIs

Ao criar uma API, Designers e Implementadores não é precisam reinventar do zero uma API que dependa de collections; já que podem utilizar as collections padrão.

Promove a reutilização de software

As estruturas de dados ligadas às interfaces de coleta padrão, e os novos algoritmos que operam em objetos que implementam interfaces são reutilizáveis.

HIERARQUIA

A Java Collections Framework (JCF) é uma arquitetura unificada para representar e manipular a coleção de dados, e fornecem um conjunto de interfaces e classes para armazenar e manipular grupos de objetos de maneira eficiente e flexível. Veremos agora algumas dessas interfaces e classes desse framework Java.

Collection:

Essa é a interface raiz e está no topo da hierarquia, e representa um grupo de elementos (objetos). Alguns tipos de collections permite valores duplicados, mas existem outros que não aceitam.

Algumas de suas sub-interfaces incluem “List”, “Set” e “Queue”.

Os seus métodos principais são: “add()”, “remove()”, “size()”, “contains()”, “iterator()”, entre outros.

List:

Representa uma coleção ordenada de elementos que podem ou não ser duplicados. A coleção ordenada é representada por uma lista sequenciada.

Com a List, os usuários tem melhor controle sobre os elementos na lista, e podem acessar esses elementos utilizando os índices, similares aos vetores.

Entre as suas implementações, estão: “ArrayList”, “LinkedList”, “Vector” e “Stack”.

Os métodos principais são: “get(int index)”, “add(int index, E element)”, “remover(int index)”, “set(int index, E element)”.

Set:

Representa uma coleção que não perite elementos duplicados.

As suas implementações são: “HashSet”, “LinkedHashSet” e “TreeSet”.

Os métodos principais são: “add(E e)”, “remove(object o)”, “size()” e “contains(Object o)”.

Queue:

Representa uma coleção usada para armazenar elementos antes do processamento.

As suas implementações são: “LinkedList” e “PriorityQueue”.

Os métodos principais são: “offer(E e)”, “poll()”, “peek()” e “remove()”.

Map:

Representa uma coleção de pares chave-valor. Embora não faça parte da collection, é frequentemente usada junto com as coleções.

As suas implementações são: “HashMap”, “LinkedHashMap”, “TreeMap”, “Hashtable”.

Os seus métodos principais são: “put(K key, V value)”, “get(Object key)”, “remove(Object key)”, “constainsKey(Object key)”, “keyset()” e “values()”.

Abaixo observamos o resumo da hierarquia em um Diagrama:

Map

LinkedList

ArrayDeque

PriorityQueue

Queue

SortedSet

HashSet

LinkedHashSet

Set

TreeSet

Stack

Vector

LinkedList

ArrayList

List

Collection

Iterable

HashSet

LinkedHashMap

Hashtable

SortedMap

TreeMap

No Diagrama acima, conseguimos ver duas interfaces, sendo elas: Collections e a Map. Vale lembrar que interfaces é um recurso que “obriga” um determinado grupo de classes a ter métodos ou prioridades em comum, para existir em um determinado contexto.

Na Collections temos as interfaces List, Set e Queue:

Na List encontramos os métodos ArrayList, LinkedList, Vector, Stack.

Em Set encontramos os métodos HashSet, LinkedHashSet, SortedSet e TreeSet (encontrado dentro do SortedSet).

Na Queue encontramos os métodos LinkedList, ArrayDeque e PriorityQueue.

A outra interface que encontramos no diagrama é o Map, que há implementado dentro de si os seguintes métodos: HashMap, LinkedHashMap, Hashtable, sortedMap e TreeMap implementado ao SortedMap.

IMPLEMENTAÇÕES

Já observamos o que são as collections e suas interfaces, também observamos alguns métodos envolvidos. Agora veremos algumas maneiras de implementar o que aprendemos.

ArrayList

Primeiro criamos uma lista, depois adicionamos valores a lista com um método da collection. Para finalizar, selecionamos o segundo valor da lista com o método “get()”, a elemento B.

Este é o melhor acesso aleatório rápido de elementos.

//Criando uma lista

List<String> list = new ArrayList<>();

// Adicionando valores a lista

list.add("A");

list.add("B");

list.add("C");

// Selecionando valores da lista

System.out.println(list.get(1));

LinkedList

Criamos uma lista que pode ser tanto List ou Deque. De maneira similar ao exemplo anterior, adicionamos os valores a lista e buscamos esses valores com o método “get()”, dessa vez o elemento C.

// Criando uma LinkedList

List<String> linkedList = new LinkedList<>();

// Adicionando valores a LinkedList

linkedList.add("A");

linkedList.add("B");

linkedList.add("C");

// Selecionando valores da lista

System.out.println(linkedList.get(2));

HashSet

Criamos a implementação de uma interface Set, e usamos uma tabela hash para armazenar os elementos. Esta interface não aceita elementos duplicados, e não garante ordem de iteração.

// Criando uma HashSet

Set<String> set = new HashSet<>();

// Adicionando valores a HashSet

set.add("A");

set.add("B");

set.add("A"); // duplicata -> não será adicionada

// Descobrindo o tamanho da HashSet

System.out.println(set.size());

HashMap

Implementamos uma interface Map que utiliza uma tabela hash E, em seguida usamos o método “put()” para inserir valores a tabela. Para finalizar, buscamos o elemento “A”, que corresponde a chave 1 da tabela.

Esta implementação permite o uso de chaves nulas e valores nulos múltiplos.

// Criando uma HashMap

Map<String, Integer> map = new HashMap<>();

// Adicionando valores a HashMap

map.put("A", 1);

map.put("B", 2);

map.put("C", 3);

// Selecionando o terceiro elemento da HashMap

System.out.println(map.get("C"));

Queue

Criamos uma interface Queue, e adicionamos três valores à lista. Em seguida selecionamos os valores a Queue, removemos um elemento e selecionamos novamente a Queue para ver os elementos que ainda restam.

// Criando uma Queue

Queue<String> queue = new LinkedList<>();

// Adicionando valores a HashMap

queue.add("Primeiro");

queue.add("Segundo");

queue.add("Terceiro");

// Selecionando os elementos da Queue

System.out.println("Elementos da Queue: " + queue);

// Removendo elementos da Queue

System.out.println("Removendo da Queue [" + queue.poll() + "]");

// Selecionando Queue depois das remoções

System.out.println("Queue depois das remoções: " + queue);

Conclusão

O Framework Collections em Java tem uma variedade de opções poderosas, que proporcionam aos desenvolvedores eficiência e flexibilidade para trabalhar. Ao escolher qual das interfaces trabalhar, leve em consideração a sua necessidade de desempenho e funcionalidade do aplicativo.

Espero que tenha gostado do conteúdo. Não deixe de fazer os exercícios para fixar na mente o que está aprendendo, e não hesite em ler este documento novamente em caso de dúvidas.

Referências

Java Collections: Como utilizar Collections

https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450

Collections (Coleções)

https://github.com/Leon4rdoalves/CookBook-Java/blob/main/15.md

Entendendo interfaces em Java

https://www.devmedia.com.br/entendendo-interfaces-em-java/25502

Interfaces em Java explicadas com exemplos

https://www.freecodecamp.org/portuguese/news/interfaces-em-java-explicadas-com-exemplos/