Sumário

[O Collection Framework 5](#_Toc130740463)

[A Interface Iterável 5](#_Toc130740464)

[A Interface Collection 5](#_Toc130740465)

[A Classe AbstractCollection 6](#_Toc130740466)

[A Interface List 6](#_Toc130740467)

[A Classe CopyOnWriteArrayList 6](#_Toc130740468)

[A Classe AbstractList 7](#_Toc130740469)

[AbstractSequentialList 7](#_Toc130740470)

[LinkedList 8](#_Toc130740471)

[Vetor 8](#_Toc130740472)

[Stack 9](#_Toc130740473)

[ArrayList 9](#_Toc130740474)

[A Interface Queue 11](#_Toc130740475)

[A Classe AbstractQueue 11](#_Toc130740476)

[ConcurrentLinkedQueue 11](#_Toc130740477)

[PriorityQueue 12](#_Toc130740478)

[A Interface BlockingQueue 13](#_Toc130740479)

[ArrayBlockingQueue 13](#_Toc130740480)

[LinkedBlockingQueue 14](#_Toc130740481)

[SynchronousQueue 15](#_Toc130740482)

[PriorityBlockingQueue 18](#_Toc130740483)

[DelayQueue 19](#_Toc130740484)

[A Interface TransferQueue 21](#_Toc130740485)

[LinkedTransferQueue 22](#_Toc130740486)

[A Interface Deque 23](#_Toc130740487)

[ArrayDeque 23](#_Toc130740488)

[ConcurrentLinkedDeque 24](#_Toc130740489)

[LinkedList 24](#_Toc130740490)

[BlockingDeque 25](#_Toc130740491)

[LinkedBlockingDeque 26](#_Toc130740492)

[A Interface Set 27](#_Toc130740493)

[A Classe AbstractSet 27](#_Toc130740494)

[A Interface SortedSet 27](#_Toc130740495)

[A Interface NavigableSet 27](#_Toc130740496)

[TreeSet 29](#_Toc130740497)

[ConcurrentSkipListSet 29](#_Toc130740498)

[HashSet 31](#_Toc130740499)

[LinkedHashSet 31](#_Toc130740500)

[CopyOnWriteArraySet 32](#_Toc130740501)

[EnumSet 34](#_Toc130740502)

[A Interface Map 35](#_Toc130740503)

[A Interface Map.Entry 35](#_Toc130740504)

[A Classe AbstractMap 38](#_Toc130740505)

[A Interface ConcurrentMap 38](#_Toc130740506)

[ConcurrentHashMap 39](#_Toc130740507)

[A Interface ConcurrentNavigableMap 40](#_Toc130740508)

[ConcurrentSkipListMap 40](#_Toc130740509)

[A Interface SortedMap 41](#_Toc130740510)

[A Interface NavigableMap 42](#_Toc130740511)

[TreeMap 42](#_Toc130740512)

[HashMap 43](#_Toc130740513)

[HashMap x HashSet 45](#_Toc130740514)

[LinkedHashMap 45](#_Toc130740515)

[WeakHashMap 46](#_Toc130740516)

[IndentityHashMap 47](#_Toc130740517)

[EnumMap 48](#_Toc130740518)

[A Classe Dictionary 49](#_Toc130740519)

[HashTable 50](#_Toc130740520)

[HashTable x HashMap 51](#_Toc130740521)

[Catálogo de Métodos 53](#_Toc130740522)

[Métodos da Interface Collection 53](#_Toc130740523)

[Métodos da Interface Iterador 54](#_Toc130740524)

[Métodos da Classe CopyOnWriteArrayList 54](#_Toc130740525)

[Métodos da Interface List 56](#_Toc130740526)

[Métodos da Classe LinkedList 58](#_Toc130740527)

[Métodos da Classe Vetor 60](#_Toc130740528)

[Métodos da Classe Stack 63](#_Toc130740529)

[Métodos da Classe ArrayList 63](#_Toc130740530)

[Métodos da Interface Queue 65](#_Toc130740531)

[Métodos da Classe ConcurrentLinkedQueue 65](#_Toc130740532)

[Métodos da Classe ProrityQueue 66](#_Toc130740533)

[Métodos da Interface Blocking Queue 66](#_Toc130740534)

[Métodos da Classe ArrayBlockingQueue 67](#_Toc130740535)

[Métodos da Classe LinkedBlockingQueue 68](#_Toc130740536)

[Métodos da Classe SynchronousQueue 69](#_Toc130740537)

[Métodos da Classe PriorityBlockingQueue 70](#_Toc130740538)

[Métodos da Classe DelayQueue 70](#_Toc130740539)

[Métodos da Interface TransferQueue 71](#_Toc130740540)

[Métodos da Classe LinkedTransferQueue 72](#_Toc130740541)

[Métodos da Interface Deque 73](#_Toc130740542)

[Métodos da Classe ArrayDeque 75](#_Toc130740543)

[Métodos da Classe ConcurrentLinkedDeque 76](#_Toc130740544)

[Métodos da Classe LinkedList 77](#_Toc130740545)

[Métodos da Interface BlockingDeque 79](#_Toc130740546)

[Métodos da Classe LinkedBlockingDeque 80](#_Toc130740547)

[Métodos da Interface Set 82](#_Toc130740548)

[Métodos da Interface SortedSet 83](#_Toc130740549)

[Métodos da Interface NavigableSet 83](#_Toc130740550)

[Métodos da Classe TreeSet 85](#_Toc130740551)

[Métodos da Classe ConcurrentSkipListSet 86](#_Toc130740552)

[Métodos da Classe HashSet 88](#_Toc130740553)

[Métodos da Classe LinkedHashSet 89](#_Toc130740554)

[Métodos da Classe CopyOnWriteArraySet 90](#_Toc130740555)

[Métodos da Classe EnumSet 91](#_Toc130740556)

[Métodos da Interface Map 92](#_Toc130740557)

[Métodos da Interface ConcurrentMap 93](#_Toc130740558)

[Métodos da Interface SortedMap 94](#_Toc130740559)

[Métodos da Interface NavigableMap 96](#_Toc130740560)

[Métodos da Interface ConcurrentNavigableMap 97](#_Toc130740561)

[Métodos da interface Map.Entry 98](#_Toc130740562)

[Métodos da Classe HashMap 98](#_Toc130740563)

[Métodos da Classe LinkedHashMap 99](#_Toc130740564)

# O Collection Framework

O Collection Framework em Java é uma estrutura de classes e interfaces que fornece uma arquitetura unificada para representar e manipular coleções de objetos. Ele fornece classes como List, Set e Map, que implementam diferentes tipos de coleções, juntamente com interfaces como Collection, Iterator e Comparator, que fornecem métodos padronizados para operações de coleção, iteração e comparação.

O Collection Framework também oferece classes de utilitários como Arrays e Collections, que fornecem métodos estáticos para trabalhar com arrays e coleções. Ele é projetado para ser flexível e extensível, permitindo que desenvolvedores criem suas próprias implementações de coleções personalizadas e as integrem com a estrutura existente.

O principal objetivo do Collection Framework é fornecer uma estrutura de dados eficiente e confiável para armazenar e manipular coleções de objetos em um programa Java. Isso ajuda a tornar a codificação mais fácil e mais rápida, evitando que os desenvolvedores precisem reinventar a roda ao implementar coleções personalizadas e operações de coleção. Além disso, o Collection Framework permite que os programas Java sejam mais escaláveis e mantenham uma estrutura de código clara e organizada.

## A Interface Iterável

A interface Iterável é a interface raiz para todas as classes da Classe coleção. A interface Collection estende a interface Iterable e, portanto, todas as subclasses da interface Collection também implementam a interface Iterable.

Ele contém apenas um método abstrato.

Iterator<T> = new iterator();

Ele retorna o iterador sobre os elementos do tipo T.

## A Interface Collection

A interface Collection é uma das interfaces fundamentais da Java Collections Framework, que é um conjunto de classes e interfaces que oferecem estruturas de dados e algoritmos para armazenar, manipular e processar coleções de objetos. Ela é a raiz da hierarquia de interfaces de coleções e define o contrato básico para uma coleção de objetos. Também contém métodos para adicionar, remover e consultar elementos da coleção, além de oferecer funcionalidades para verificar se a coleção contém um determinado elemento, obter o tamanho da coleção, verificar se a coleção está vazia, e obter uma representação em array da coleção.

Fora isso, a interface Collection define uma série de métodos padrão que são implementados por suas subinterfaces, como List, Set e Queue. Esses métodos incluem operações para realizar operações em massa, como adicionar todos os elementos de outra coleção, remover todos os elementos que correspondem a uma determinada condição e reter apenas os elementos que correspondem a uma determinada condição.

A interface Collection também define o comportamento padrão para o método equals(), que é usado para comparar coleções. Duas coleções são consideradas iguais se contêm os mesmos elementos na mesma ordem.

No geral, a interface Collection é um componente fundamental da Java Collections Framework e é usada extensivamente em muitas aplicações Java. Ela fornece uma base consistente para trabalhar com coleções de objetos, independentemente do tipo específico de coleção usada.

### A Classe AbstractCollection

A classe AbstractCollection é uma classe abstrata que fornece uma implementação padrão para alguns dos métodos da interface Collection e pode ser usada como base para a implementação de classes que desejam fornecer uma implementação personalizada de uma coleção de objetos. Basicamente ela prove implementações básicas das interfaces da coleção principal, para minimizar o esforço necessário para implementá-las. Há também as classes AbstractSet, AbstractList, AbstractSequentialList e AbstractMap que estendem esta classe.

#### A Interface List

A interface List é a interface filha da interface Collection. Ele inibe uma estrutura de dados do tipo list na qual podemos armazenar a coleção ordenada de objetos. Podendo ela ter valores duplicados.

A interface List é implementada pelas classes ArrayList, LinkedList, Vector e Stack.

Para instanciar a interface List, devemos usar:

List <tipo de dado> list1 = new ArrayList();

List <tipo de dado> list2 = new LinkedList();

List <tipo de dado> list3 = new Vector();

List <tipo de dado> list4 = new Stack();

Existem vários métodos na interface List que podem ser usados ​​para inserir, excluir e acessar os elementos da lista.

As classes que implementam a interface List são fornecidas abaixo.

##### A Classe CopyOnWriteArrayList

Uma variante thread-safe da ArrayList na qual todas as operações mutativas (adicionar, definir e assim por diante) são implementadas fazendo uma nova cópia da matriz subjacente.

Normalmente, seu uso é muito caro, mas pode ser mais eficiente do que alternativas quando as operações de travessia superam amplamente as mutações e é útil quando você não pode ou não deseja sincronizar as travessias, mas precisa impedir a interferência entre threads simultâneos. O método iterador de estilo "instantâneo" usa uma referência ao estado da matriz no ponto em que o iterador foi criado. Essa matriz nunca muda durante o tempo de vida do iterador, portanto, a interferência é impossível e o iterador tem a garantia de não lançar ConcurrentModificationException. O iterador não refletirá adições, remoções ou alterações na lista desde que o iterador foi criado. As operações de alteração de elemento nos próprios iteradores (remover, definir e adicionar) não são suportadas. Esses métodos lançam UnsupportedOperationException.

Exemplo:

public class ConcurrentDemo extends Thread {

static CopyOnWriteArrayList<String> l

= new CopyOnWriteArrayList<String>();

public void run()

{

// Filho do thread tentando adicionar

// um elemento ao objeto coleção

l.add("D");

}

public static void main(String[] args) throws InterruptedException {

l.add("A");

l.add("B");

l.add("c");

// Criamos um filho do thread

// que ira modifical a ArrayList l

ConcurrentDemo t = new ConcurrentDemo();

t.start();

Thread.sleep(1000);

// Agora será iterado pela ArrayList

// para adquirir a exceção

Iterator itr = l.iterator();

while (itr.hasNext()) {

String s = (String) itr.next();

System.out.println(s);

Thread.sleep(1000);

}

System.out.println(l);

}

}

##### A Classe AbstractList

A classe AbstractList é uma classe abstrata que implementa parcialmente a interface List. Essa classe fornece uma implementação padrão para vários métodos da interface List, incluindo métodos de acesso e modificação de elementos, bem como métodos de busca e ordenação de elementos. A classe AbstractList é frequentemente usada como base para a implementação de classes que desejam fornecer uma lista personalizada de objetos.

###### AbstractSequentialList

A classe AbstractSequentialList é uma classe abstrata que estende a classe AbstractList e implementa parcialmente a interface List. Ela fornece uma implementação padrão para métodos específicos de lista sequencial, como o método get(), o método listIterator() e o método add().

A lista sequencial é uma lista de elementos em que cada elemento tem um sucessor e um antecessor definidos, exceto pelo primeiro e pelo último elemento. A classe AbstractSequentialList fornece uma estrutura básica para implementação de uma lista sequencial de objetos, onde os elementos são organizados em uma sequência.

Ela implementa o método listIterator(), que retorna um iterador para percorrer a lista sequencial em ordem. Além disso, a classe implementa o método add(), que insere um elemento em uma posição especificada na lista sequencial, e o método get(), que retorna o elemento em uma determinada posição.

É frequentemente usada como base para a implementação de classes que desejam fornecer uma lista sequencial personalizada de objetos. As classes que estendem a classe AbstractSequentialList precisam implementar os métodos abstratos da classe AbstractList e podem fornecer implementações adicionais para métodos específicos da lista sequencial, como o método set() e o método remove().

LinkedList

Uma Linked List em Java é uma implementação da estrutura de dados de lista encadeada, que é uma sequência de elementos onde cada elemento é um objeto que contém um valor e um ponteiro para o próximo elemento da lista. Na LinkedList, a manipulação é rápida porque nenhum deslocamento é necessário.

Exemplo:

class TestJavaCollection2{

static void main(String args[]){

LinkedList<String> al = new LinkedList<String>();

al.add("Ravi");

al.add("José");

al.add("Ravi");

al.add("Rose");

Iterator<String> al = list.iterator();

while(itr.hasNext()){

System.out.println(al.next());

}

}

}

**Output:**

Ravi

José

Ravi

Rose

###### Vetor

A Classe Vetor usa uma array dinâmica para armazenar os elementos de dados. É semelhante ao ArrayList. No entanto, é sincronizado e contém muitos métodos que não fazem parte do framework Collection.

Exemplo:

class TestJavaCollection3{

static void main(String args[]){

Vector<String> v = new Vector<String>();

v.add("João");

v.add("Maria");

v.add("Armando");

v.add("Marcos");

Iterator<String> itr = v.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Output:**

João

Maria

Armando

Marcos

Stack

O stack é a subclasse de Vector. Ele implementa a estrutura de dados (LIFO) last-in-first-out. Ele contém todos os métodos da classe Vector e também fornece seus métodos como push(), peek(), push(object o), que define suas propriedades.

Exemplo:

class TestJavaCollection4{

static void main(String args[]){

Stack<String> stack = new Stack<String>();

stack.push("Thema");

stack.push("Wine");

stack.push("Siim");

stack.push("Gábor");

stack.push("Thema");

stack.pop();

Iterator<String> itr=stack.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Output:**

Thema

Wine

Siim

Gábor

###### ArrayList

As ArraysList funcionam como uma array comum, tendo como sua excêntrica diferença a dinamicidade, ou seja, a capacidade de remover e adicionar elementos, o que se fosse feito com uma array comum teria de ser criado uma nova. Sua sintaxe tem uma pequena diferença:

Além da utilização do import Java.util.ArrayList;

ArrayList<Tipo de Dado> Nome da Variável = new ArrayList<Tipo de Dado>();

Exemplo: ArrayList<Integer> números = new ArrayList<Integer>();

Para adicionar elementos, é utilizado o método add:

Exemplo: ArrayList<String> nomes = new ArrayList<String>();

nomes.add(“Jack”);

Para remover elementos, é utilizado o método remove, no qual o argumento utilizado é a posição do elemento:

Exemplo: ArrayList<Integer> i = new ArrayList<Integer>();

i.add(1);

i.add(5);

i.add(20);

i.remove(1);

//aqui será removido o valor 5 que está no índice 1

Em Arrays comuns é utilizado o metodo length para se obter o tamanho, já para ArrayLists é utilizado o método size().

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

**Output:**

2

O principal motivo para utilizar uma ArrayList é pela sua dinamicidade, pois tirando isso toda a sua funcionalidade é da Classe array comum.

Exemplo:

class TestJavaCollection1{

static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();//criando a arraylist

list.add("Ravi"); //Adicionando os objetos

list.add("José");

list.add("Ravi");

list.add("Rose");

// Atravessando a arraylist com um iterador

Iterator<String> itr = list.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Output:**

Ravi

José

Ravi

Rose

#### A Interface Queue

A interface Queue mantém a ordem do primeiro a entrar, e do primeiro a sair. Pode ser definido como uma lista ordenada que é usada para conter os elementos que estão prestes a serem processados. Existem várias classes como PriorityQueue, Deque e ArrayDeque que implementam a interface Queue.

##### A Classe AbstractQueue

Essa classe fornece implementações básicas de algumas operações de Fila. As implementações nesta classe são apropriadas quando a sua base não permite elementos nulos. Os métodos add, remove e element são baseados em offer, poll e peek, respectivamente, mas lançam exceções em vez de indicar falha por meio de retornos falsos ou nulos. Uma implementação de Queue que estenda esta classe deve definir minimamente um método Queue.offer(E) que não permita a inserção de elementos nulos, juntamente com os métodos Queue.peek(), Queue.poll(), Collection.size() e Collection.iterator(). Normalmente, métodos adicionais também serão substituídos. Se esses requisitos não puderem ser atendidos, considere a subclasse AbstractCollection.

Vale ressaltar que a interface BlockingQueue e TransferQueue não estendem essa classe.

###### ConcurrentLinkedQueue

Ele é utilizado com o intuito de implementar a interface Queue simultaneamente com a ajuda da LinkedList. É uma implementação thread-safe ilimitada que insere elementos no final de maneira FIFO (entra primeiro, sai primeiro). Pode ser utilizado quando uma Fila ilimitada é compartilhada entre muitos encadeamentos. Esta classe não permite elementos nulos. Os iteradores são fracamente consistentes. Essa classe implementa todos os métodos adicionais da interface Queue e Iterator.

Exemplo:

class ConcurrentLinkedQueue{

public static void main(String[] args)

{

// Criando uma ConcurrentLinkedQueue

// Utilizando o construtor ConcurrentLinkedQueue()

ConcurrentLinkedQueue<Integer>

clq = new ConcurrentLinkedQueue<Integer>();

clq.add(12);

clq.add(70);

clq.add(1009);

clq.add(475);

// Mostrando a LinkedQueue existente

System.out.println("ConcurrentLinkedQueue: "

+ clq);

// Criando uma ConcurrentLinkedQueue

// Utilizando o construtor ConcurrentLinkedQueue(Collection c)

ConcurrentLinkedQueue<Integer>

clq1 = new ConcurrentLinkedQueue<Integer>(clq);

// Mostrando a LinkedQueue existente

System.out.println("ConcurrentLinkedQueue1: "

+ clq1);

}

}

**Output:**

ConcurrentLinkedQueue: [12, 70, 1009, 475]

ConcurrentLinkedQueue1: [12, 70, 1009, 475]

###### PriorityQueue

A classe PriorityQueue implementa a interface Queue. Ele contém os elementos ou objetos que devem ser processados ​​por suas prioridades. A PriorityQueue não permite que valores nulos sejam armazenados na fila.

Exemplo:

class TestJavaCollection5{

static void main(String args[]){

PriorityQueue<String> queue = new PriorityQueue<String>();

queue.add("Thema");

queue.add("Rodrigo");

queue.add("Zion");

queue.add("Charles");

System.out.println("primeiro:"+queue.element());

System.out.println("primeiro:"+queue.peek());

System.out.println("iterando pelos elementos da fila:");

Iterator<String> itr = queue.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

queue.remove();

queue.poll();

System.out.println("depois de ter sido removido dois elementos da fila:");

Iterator<String> itr2=queue.iterator();

while(itr2.hasNext()){

System.out.println(itr2.next());

}

}

}

**Output:**

primeiro:Charles

primeiro:Charles

iterando pelos elementos da fila:

Charles

Rodrigo

Zion

Thema

depois de ter sido removido dois elementos da fila:

Thema

Zion

###### A Interface BlockingQueue

Antes de pular diretamente para o tópico, é melhor relembrar o que é realmente uma Fila. Fila é uma lista ordenada de objetos em que as inserções ocorrem no final da lista e a exclusão de elementos ocorre no início. Portanto, também é dito que a fila é baseada no princípio FIFO (First-In-First-Out).

BlockingQueue é uma fila que também oferece suporte a operações que esperam que a fila fique não vazia quando estamos tentando recuperar um elemento e esperamos que o espaço fique vazio quando um elemento deve ser inserido na fila. A fila é bloqueada quando um pacote está em trânsito entre ela e seu vizinho downstream (na maioria das vezes se a fila estiver ocupada). Uma fila bloqueada permanecerá bloqueada enquanto seu link downstream estiver ocupado e a fila tiver pelo menos um pacote para enviar.

A BlockingQueue tem dois tipos:

1. Fila ilimitada: A capacidade da fila de bloqueio será definida como Integer.MAX\_VALUE. No caso da Classe fila de bloqueio ilimitada, a fila nunca será bloqueada porque pode crescer para um tamanho muito grande. Quando você adiciona elementos, seu tamanho aumenta.

Sintaxe:

BlockingQueue blockingQueue = new LinkedBlockingQueue();

2. Fila Limitada: O segundo tipo de fila é a fila limitada. No caso da Classe fila limitada você pode criar uma fila passando a capacidade da fila no construtor de filas:

Sintaxe:

// Cria uma Blocking Queue com capacidade 5

BlockingQueue blockingQueue = new ArrayBlockingQueue(5);

ArrayBlockingQueue

ArrayBlockingQueue é uma Blocking Queue limitada que ordena o elemento em FIFO (First-In-First-Out). Nesta fila, novos elementos são inseridos no final e os elementos são recuperados do início desta fila.

Exemplo:

class JavaArrayBlockingQueueExample1 {

public static void main(String[] args) throws InterruptedException {

int capacidade = 10;

BlockingQueue<String> queue = new

ArrayBlockingQueue<String>(capacidade);

//add() method inserts the elements at the tail of this queue

queue.add("Nay");

queue.add("Rahul");

queue.add("Rita");

queue.add("Cindel");

//offer() este método insere os elementos no final des fila se houver espaço

queue.offer("Villum");

//put() este método insere os elementos no final des fila espereando, se não houver espaço

queue.put("Mellan");

// forEach() este método executa o iterador para cada elemento.

System.out.println("Elementos: ");

for (String xyz : queue){

System.out.println(xyz);

}

/\*o método contains() irá retornar um boolean se o valor especificado estiver presente nesta fila\*/

System.out.println("queue.contains(nay) irá retornar "+queue.contains("Nay"));

}

}

**Output:**

Elementos:

Nay

Rahul

Rita

Cindel

Villum

Mellan

queue.contains(reema) irá retornar true

LinkedBlockingQueue

O LinkedBlockingQueue é uma fila de bloqueio opcionalmente limitada com base em nós vinculados. Isso significa que o LinkedBlockingQueue pode ser limitado, se sua capacidade for fornecida, senão o LinkedBlockingQueue será ilimitado. A capacidade pode ser fornecida como um parâmetro para o construtor de LinkedBlockingQueue. Esta fila ordena os elementos FIFO (first-in-first-out). Significa que o cabeçalho desta fila é o elemento mais antigo dos elementos presentes nesta fila. A cauda desta fila é o elemento mais novo dos elementos desta fila. Os elementos recém inseridos são sempre inseridos no final da fila, e as operações de recuperação da fila obtêm elementos no início da fila. As filas vinculadas geralmente têm rendimento mais alto do que as filas baseadas em array, mas desempenho menos previsível na maioria dos aplicativos simultâneos.

A capacidade, se não especificada, é igual a Integer.MAX\_VALUE. Os nós vinculados são criados dinamicamente a cada inserção, até que a capacidade da fila não seja preenchida. Essa classe e seu iterador implementam todos os métodos opcionais das interfaces Collection e Iterator. É um membro do Java Collections Framework.

Exemplo:

public class LinkedBlockingQueueDemo {

public static void main(String[] args)

{

// crinado um objeto da LinkedBlockingQueue

// utilizando o construtor da LinkedBlockingQueue()

LinkedBlockingQueue<Integer> lbq

= new LinkedBlockingQueue<Integer>();

// adicionando números

lbq.add(1);

lbq.add(2);

lbq.add(3);

lbq.add(4);

lbq.add(5);

// imprimindo na tela

System.out.println("LinkedBlockingQueue:" + lbq);

}

}

**Output:**

LinkedBlockingQueue:[1, 2, 3, 4, 5]

SynchronousQueue

A palavra "Synchronous" na classe java.util.concurrent.SynchronousQueue não deve ser confundida com "synchronized". Em vez disso, significa que cada operação de enfileiramento deve aguardar uma operação de desenfileiramento correspondente e cada operação de desenfileiramento deve aguardar uma operação de enfileiramento.

Ela é uma fila de bloqueio na qual cada operação de inserção deve aguardar uma operação de remoção correspondente por outro thread e vice-versa. Uma fila síncrona não possui nenhuma capacidade interna, nem mesmo capacidade da Classe . Você não pode espiar uma fila síncrona porque um elemento só está presente quando você tenta removê-lo; você não pode inserir um elemento (usando qualquer método) a menos que outra thread esteja tentando removê-lo; você não pode iterar porque não há nada para iterar. O início da fila é o elemento que o primeiro encadeamento de inserção na fila está tentando adicionar à fila; se não houver tal encadeamento na fila, nenhum elemento estará disponível para remoção e poll () retornará nulo. Para fins de outros métodos Collection (por exemplo, contains), um SynchronousQueue atua como uma coleção vazia. Esta fila não permite elementos nulos.

As filas síncronas são semelhantes aos canais de encontro usados ​​em CSP e Ada. Eles são adequados para projetos de handoff, nos quais um objeto em execução em um thread deve sincronizar com um objeto em execução em outro thread para entregar a ele alguma informação, evento ou tarefa.

Essa classe oferece suporte a uma política de imparcialidade opcional para solicitar encadeamentos de produtores e consumidores em espera. Por padrão, essa ordem não é garantida. No entanto, uma fila construída com imparcialidade definida como true concede acesso aos threads na ordem FIFO.

Essa classe e seu iterador implementam todos os métodos opcionais das interfaces Collection e Iterator.

Essa classe é membro do Java Collections Framework.

Exemplo:

public class SynchronousQAPI<E> {

public SynchronousQueue<E> synchronousQ;

// criamos uma SynchronousQueue sem uma política justa

public SynchronousQAPI()

{

synchronousQ = new SynchronousQueue<E>();

}

// criamos um SynchronousQueue com política justa

public SynchronousQAPI(boolean fair)

{

synchronousQ = new SynchronousQueue<E>();

}

// SynchronousQueue tem 2 operações suportadas put() e

// take() Então, vamos implementar apenas esses métodos

// Método put(): Insere o elemento no final da fila

// e costumava esperar até que a fila estivesse cheia.

public void put(E e) throws InterruptedException

{

synchronousQ.put(e);

}

// método take(): retorna o elemento no início da fila

public E take() throws InterruptedException

{

return synchronousQ.take();

}

// Implementação do Put Thread (produtor)

class Put implements Runnable {

@SuppressWarnings("rawtypes")

BlockingQueue SynchronousQueue;

@SuppressWarnings("rawtypes")

public Put(BlockingQueue q)

{

this.SynchronousQueue = q;

}

@SuppressWarnings("unchecked")

@Override

public void run()

{

try {

// colocando os dados

SynchronousQueue.put(1);

System.out.println(

"1 adicionado a fila síncrona.");

Thread.sleep(1000);

}

catch (InterruptedException e) {

e.printStackTrace();

}

}

}

class Take implements Runnable {

@SuppressWarnings("rawtypes")

BlockingQueue SynchronousQueue;

@SuppressWarnings("rawtypes")

public Take(BlockingQueue q)

{

this.SynchronousQueue = q;

}

@Override public void run()

{

try {

// retirando os dados inseridos anteriormente this.SynchronousQueue.take();

System.out.println(

"1 removido da fila síncrona.");

}

catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public static void main(String[] args)

throws InterruptedException

{

SynchronousQAPI<Integer> synchronousQueue

= new SynchronousQAPI<Integer>();

new Thread(new SynchronousQAPI<>().new Put(

synchronousQueue.synchronousQ))

.start();

new Thread(new SynchronousQAPI<>().new Take(

synchronousQueue.synchronousQ))

.start();

}

}

**Output:**

1 adicionado à fila síncrona.

1 removido da fila síncrona.

PriorityBlockingQueue

A PriorityBlockingQueue é uma fila de bloqueio ilimitada que usa as mesmas regras de ordenação da classe PriorityQueue e fornece operações de recuperação de bloqueio. Como é ilimitado, adicionar elementos às vezes pode falhar devido ao esgotamento de recursos, resultando em OutOfMemoryError. Esta classe não permite elementos nulos.

A classe PriorityBlockingQueue e seu iterador implementam todos os métodos opcionais das interfaces Collection e Iterator.

Não há garantia de que o Iterator fornecido no método iterator() e o Spliterator fornecido no método spliterator() atravessem os elementos de PriorityBlockingQueue em qualquer ordem específica. Para percurso ordenado, use Arrays.sort(pq.toArray()). Além disso, o método drainTo() pode ser usado para remover alguns ou todos os elementos em ordem de prioridade e colocá-los em outra coleção.

As operações nesta classe não garantem a ordem dos elementos com igual prioridade. Se for necessário impor uma ordem, defina classes ou comparadores customizados que usam uma chave secundária para desvincular os valores de prioridade primária.

Essa classe é membro do Java Collections Framework.

Exemplo:

public class GFG {

public static void main(String[] args)

{

// criando um objeto da PriorityBlockingQueue

// utilizando o construtor da PriorityBlockingQueue()

PriorityBlockingQueue<Integer> pbq

= new PriorityBlockingQueue<Integer>();

// adicionando os números

pbq.add(1);

pbq.add(2);

pbq.add(3);

pbq.add(4);

pbq.add(5);

// imprimindo a fila

System.out.println("PriorityBlockingQueue:" + pbq);

}

}

**Output:**

PriorityBlockingQueue:[1, 2, 3, 4, 5]

DelayQueue

A classe DelayQueue é um membro do Java Collections Framework. Pertence ao pacote java.util.concurrent. DelayQueue implementa a interface BlockingQueue. DelayQueue é uma fila de prioridade especializada que ordena elementos com base em seu tempo de atraso. Isso significa que apenas os elementos podem ser retirados da fila cujo tempo expirou.

O cabeçalho DelayQueue contém o elemento que expirou no menor tempo. Se nenhum atraso tiver expirado, não haverá cabeça e a pesquisa retornará nula. DelayQueue aceita apenas os elementos que pertencem a uma classe do tipo Delayed ou que implementam a interface java.util.concurrent.Delayed. O DelayQueue bloqueia os elementos internamente até que um certo atraso tenha expirado. DelayQueue implementa o método getDelay(TimeUnit.NANOSECONDS) para retornar o tempo de atraso restante. A instância TimeUnit passada para o método getDelay() é um Enum que informa em qual unidade de tempo o atraso deve ser retornado. A enumeração TimeUnit pode levar DIAS, HORAS, MINUTOS, SEGUNDOS, MILISEGUNDOS, MICROSSEGUNDOS, NANOSEGUNDOS. Esta fila não permite elementos nulos. Essa classe e seu iterador implementam todos os métodos opcionais das interfaces Collection e Iterator. O Iterator fornecido no método iterator() não tem garantia de percorrer os elementos do DelayQueue em qualquer ordem específica.

Exemplo:

// O DelayObject para DelayQueue

// Deve implementar Delayed e

// seu método getDelay() e compareTo()

class DelayObject implements Delayed {

private String name;

private long time;

// Constructor of DelayObject

public DelayObject(String name, long delayTime)

{

this.name = name;

this.time = System.currentTimeMillis()

+ delayTime;

}

// Implementando o método getDelay() do Delayed

@Override

public long getDelay(TimeUnit unit)

{

long diff = time - System.currentTimeMillis();

return unit.convert(diff, TimeUnit.MILLISECONDS);

}

// Implementando o método compareTo() do Delayed

@Override

public int compareTo(Delayed obj)

{

if (this.time < ((DelayObject)obj).time) {

return -1;

}

if (this.time > ((DelayObject)obj).time) {

return 1;

}

return 0;

}

// Implementando o método toString() do Delayed

@Override

public String toString()

{

return "\n{"

+ "name=" + name

+ ", time=" + time

+ "}";

}

}

// Classe principal

public class GFG {

public static void main(String[] args)

throws InterruptedException

{

// cria objeto de DelayQueue

// usando o construtor DelayQueue()

BlockingQueue<DelayObject> DQ = new DelayQueue<DelayObject>();

// Adicionar números ao final do DelayQueue

DQ.add(new DelayObject("A", 1));

DQ.add(new DelayObject("B", 2));

DQ.add(new DelayObject("C", 3));

DQ.add(new DelayObject("D", 4));

// imprimindo a DelayQueue

System.out.println("DelayQueue: "

+ DQ);

// cria objeto de DelayQueue

// usando o construtor DelayQueue(Collection c)

BlockingQueue<DelayObject> DQ2 = new DelayQueue<DelayObject>(DQ);

// imprimindo a DelayQueue

System.out.println("DelayQueue: "

+ DQ2);

}

}

**Output:**

DelayQueue: [

{name=A, time=1543472836003},

{name=B, time=1543472836004},

{name=C, time=1543472836005},

{name=D, time=1543472836006}]

DelayQueue: [

{name=A, time=1543472836003},

{name=B, time=1543472836004},

{name=C, time=1543472836005},

{name=D, time=1543472836006}]

A Interface TransferQueue

A interface TransferQueue é um membro do Java Collections Framework. Foi introduzido no JDK 1.7, pertence ao pacote java.util.concurrent. O TransferQueue é um BlockingQueue no qual um thread de envio (produtor) pode esperar que o thread de recebimento (consumidores) receba elementos. TransferQueue é usado em aplicativos de passagem de mensagens. Existem dois aspectos em que a mensagem será passada do encadeamento do produtor para o encadeamento do consumidor.

* put(E e): Este método é usado se o produtor deseja enfileirar elementos sem esperar por um consumidor. No entanto, ele espera até que o espaço fique disponível se a fila estiver cheia.
* transfer(E e): Este método geralmente é usado para transferir um elemento para um thread que está esperando para recebê-lo, se não houver nenhum thread esperando, ele aguardará até que um thread chegue ao estado de espera assim que o thread em espera chegar elemento será transferido para ele.

Uma TransferQueue também pode ser consultada, por meio de hasWaitingConsumer(), se há threads esperando por itens, o que é uma analogia inversa a uma operação peek.

Exemplo:

public class TransferQueueDemo {

public static void main(String[] args)

throws InterruptedException

{

// criando o objeto TransferQueue

// utilizando o construtor LinkedTransferQueue()

TransferQueue<Integer> TQ = new LinkedTransferQueue<Integer>();

// adicionando números no final da fila

TQ.add(7855642);

TQ.add(35658786);

TQ.add(5278367);

TQ.add(74381793);

// Imprimindo a fila

System.out.println("Queue1: " + TQ);

// criando o objeto TransferQueue

// utilizando o construtor LinkedTransferQueue(Collection c)

TransferQueue<Integer> TQ2

= new LinkedTransferQueue<Integer>(TQ);

// Imprimindo a fila

System.out.println("Queue2: " + TQ2);

}

}

**Output:**

Queue1: [7855642, 35658786, 5278367, 74381793]

Queue2: [7855642, 35658786, 5278367, 74381793]

LinkedTransferQueue

A classe LinkedTransferQueue em Java faz parte do Java Collection Framework. Foi introduzido no JDK 1.7 e pertence ao pacote java.util.concurrent. Ele implementa o TransferQueue e fornece uma funcionalidade ilimitada baseada em nós vinculados. Os elementos no LinkedTransferQueue são ordenados em ordem FIFO, com a cabeça apontando para o elemento que está na Fila há mais tempo e a cauda apontando para o elemento que está na fila há menos tempo. Devido à sua natureza assíncrona, size() percorre toda a coleção, portanto, não é uma operação de tempo O(1). Também pode fornecer um tamanho impreciso se esta coleção for modificada durante a travessia. Operações em massa como addAll, removeAll, reterAll, containsAll, equals e toArray não têm garantia de serem executadas atomicamente. Por exemplo, um iterador operando simultaneamente com uma operação addAll pode observar apenas alguns dos elementos adicionados.

LinkedTransferQueue é utilizado para aplicativos de passagem de mensagens. Existem dois aspectos em que a mensagem será passada do encadeamento do produtor para o encadeamento do consumidor.

put(E e): Este método é usado se o produtor deseja enfileirar elementos sem esperar por um consumidor. No entanto, ele espera até que o espaço fique disponível se a fila estiver cheia.

transfer(E e): Este método geralmente é usado para transferir um elemento para um thread que está esperando para recebê-lo, se não houver nenhum thread esperando, ele aguardará até que um thread chegue ao estado de espera assim que o thread em espera chegar elemento será transferido para ele.

Exemplo:

public class LinkedTransferQueueDemo {

public static void main(String[] args)

throws InterruptedException

{

// criando um objeto da LinkedTransferQueue

// utilizando o construtor LinkedTransferQueue()

LinkedTransferQueue<Integer> LTQ

= new LinkedTransferQueue<Integer>();

// Adicionando elementos ao final da LinkedTransferQueue

LTQ.add(7855642);

LTQ.add(35658786);

LTQ.add(5278367);

LTQ.add(74381793);

// imprimindo a fila

System.out.println("Linked Transfer Queue1: " + LTQ);

// criando um objeto da LinkedTransferQueue

// utilizando o construtor LinkedTransferQueue(Collection c)

LinkedTransferQueue<Integer> LTQ2

= new LinkedTransferQueue<Integer>(LTQ);

// imprimindo a fila

System.out.println("Linked Transfer Queue2: " + LTQ2);

}

}

**Output:**

Linked Transfer Queue1: [7855642, 35658786, 5278367, 74381793]

Linked Transfer Queue2: [7855642, 35658786, 5278367, 74381793]

##### A Interface Deque

A interface Deque estende a interface Queue. No Deque, podemos remover e adicionar os elementos de ambos os lados. Deque significa uma fila dupla que nos permite realizar as operações em ambas as extremidades.

###### ArrayDeque

A classe ArrayDeque implementa a interface Deque. Facilita a utilização do Deque. Ao contrário da fila, podemos adicionar ou excluir os elementos de ambas as extremidades.

ArrayDeque é mais rápido que ArrayList e Stack e não tem restrições de capacidade.

Exemplo:

class TestJavaCollection6{

static void main(String[] args) {

//Criando deque e adicionando seus elementos

Deque<String> deque = new ArrayDeque<String>();

deque.add("Gautam");

deque.add("Karan");

deque.add("Nay");

//Passando pelos elementos

for (String str : deque) {

System.out.println(str);

}

}

}

**Output:**

Gautam

Karan

Nay

###### ConcurrentLinkedDeque

Um deque simultâneo ilimitado baseado em nós vinculados. As operações simultâneas de inserção, remoção e acesso são executadas com segurança em vários segmentos. Um ConcurrentLinkedDeque é uma escolha apropriada quando muitos threads compartilham o acesso a uma coleção comum. Como a maioria das outras implementações de coleção simultâneas, essa classe não permite o uso de elementos nulos.

Iteradores e divisores são fracamente consistentes.

Esteja ciente de que, ao contrário da maioria das coleções, o método size NÃO é uma operação de tempo constante. Devido à natureza assíncrona desses deques, determinar o número atual de elementos requer uma travessia dos elementos e, portanto, pode relatar resultados imprecisos se essa coleção for modificada durante a travessia. Além disso, as operações em massa addAll, removeAll, reterAll, containsAll, equals e toArray não têm garantia de serem executadas atomicamente. Por exemplo, um iterador operando simultaneamente com uma operação addAll pode exibir apenas alguns dos elementos adicionados.

Esta classe e seu iterador implementam todos os métodos opcionais das interfaces Deque e Iterator.

Efeitos de consistência de memória: como em outras coleções simultâneas, as ações em um thread antes de colocar um objeto em um ConcurrentLinkedDeque acontecem antes das ações subsequentes ao acesso ou remoção desse elemento do ConcurrentLinkedDeque em outro thread.

Exemplo:

public class Example {

public static void main(String[] args) {

Deque<Integer> deque = new ConcurrentLinkedDeque<>();

deque.addFirst(1);

deque.addLast(2);

int first = deque.pollFirst();

int last = deque.pollLast();

System.out.println("Primeiro: " + first + ", Ultimo: " + last);

}

}

**Output:**

First: 1, Last: 2

###### LinkedList

A implementação da LinkedList usando a interface Deque é uma forma de criar uma lista ligada que pode ser usada como uma fila. Para fazer isso, é preciso criar uma nova classe que implementa a interface Deque e usa a LinkedList como uma estrutura interna de dados para armazenar os elementos da fila.

Essa nova classe, que é chamada de LinkedListDeque, utiliza os métodos da interface Deque para inserir e remover elementos tanto no início quanto no final da fila. Isso inclui métodos como addFirst(), offerFirst(), removeFirst(), pollFirst(), getFirst() e peekFirst() para inserir e remover elementos do início da fila, e addLast(), offerLast(), removeLast(), pollLast(), getLast() e peekLast() para inserir e remover elementos do final da fila.

No construtor da classe LinkedListDeque, é criada uma nova LinkedList para armazenar os elementos da fila. Depois, a implementação dos métodos da interface Deque é feita de forma a delegar a chamada dos métodos da LinkedList interna, para que as operações de inserção, remoção e acesso de elementos sejam realizadas na fila.

Por exemplo, se alguém quiser adicionar um elemento no início da fila usando o método addFirst(T item) da interface Deque, basta chamar esse método na classe LinkedListDeque e ela vai chamar o método addFirst(T item) da LinkedList interna para adicionar o elemento no início da fila. Isso também acontece com os outros métodos da interface Deque.

Essa abordagem de implementação é muito útil, porque a interface Deque é uma abstração que pode ser usada em conjunto com outras classes ou interfaces que também utilizam a interface Deque. Isso torna o código mais modular, fácil de manter e reutilizável em diferentes contextos.

###### BlockingDeque

A interface BlockingDeque faz parte do Java Collections Framework. Ele recebe esse nome porque bloqueia operações ilegais, como inserção em uma fila cheia ou exclusão da Classe fila vazia, todas essas propriedades são incorporadas à estrutura dessa interface. Como é um deque (fila de duas pontas), ou seja, as operações de inserção, exclusão e travessia podem ser executadas de ambas as extremidades. O BlockingDeque é uma interface, então não podemos declarar nenhum objeto com ele.

Exemplo:

public class BlockingDequeExample {

public static void main(String[] args) {

// Instancia um objeto utilizando LinkedBlockingDeque nomeada lbdq

BlockingDeque<Integer> lbdq = new LinkedBlockingDeque<Integer>();

// adiciona elementos utilizando o método add()

lbdq.add(134);

lbdq.add(245);

lbdq.add(23);

lbdq.add(122);

lbdq.add(90);

// Cria um iterador para atravessar o deque

Iterator<Integer> lbdqIter = lbdq.iterator();

// imprimindo os elementos da lbdq no console

System.out.println("A LinkedBlockingDeque lbdq contém:");

for(int i = 0; i<lbdq.size(); i++)

{

System.out.print(lbdqIter.next() + " ");

}

// Remove 23 e mostra a mensagem apropriada se a

// operação for sucedida

if(lbdq.remove(23))

{

System.out.println("\n\nO elemento 23 foi removido ");

}

else

{

System.out.println("\n\nNenhum elementos foi encontrado ");

}

// Imprime os elementos da lbdq sem utilizar um iterador

System.out.println("\nA LinkedBlockingDeque lbdq"+

" depois da operação de remoção contém:");

System.out.println(lbdq);

}

}

**Output:**

A LinkedBlockingDeque lbdq contém:

134 245 23 122 90

O elemento 23 foi removido

A LinkedBlockingDeque lbdq depois da operação de remoção contém:

[134, 245, 122, 90]

LinkedBlockingDeque

A classe LinkedBlockingDeque em Java faz parte do Java Collection Framework. Foi introduzido no JDK 1.6 e pertence ao pacote java.util.concurrent. É um Deque (Doubly Ended Queue) que bloqueia um thread se esse thread tentar retirar elementos dele enquanto o Deque estiver vazio. Ele implementa o BlockingDeque e fornece uma funcionalidade opcionalmente limitada com base em nós vinculados. Essa limitação opcional é fornecida passando o tamanho necessário no construtor e ajuda a evitar o desperdício de memória. Quando não especificada, a capacidade é tomada por padrão como Integer.MAX\_VALUE. Essa classe e seu iterador implementam todos os métodos opcionais das interfaces Collection e Iterator. A implementação fornecida pelo LinkedBlockingDeque é thread-safe. Todos os métodos de enfileiramento na classe obtêm seus efeitos atomicamente usando ReentrantLock internamente.

A classe LinkedBlockingDeque em Java é uma implementação thread-safe e concorrente da interface Deque que usa uma lista encadeada para armazenar seus elementos. Um LinkedBlockingDeque pode ser usado como uma pilha ou fila tradicional, dependendo do método usado para inserir e remover elementos.

Exemplo:

public class LinkedBlockingDequeExample {

public static void main(String[] args) throws InterruptedException {

LinkedBlockingDeque<Integer> deque = new LinkedBlockingDeque<>(5);

// Adicionando elementos ao deque

deque.offerFirst(1);

deque.offerLast(2);

deque.offerFirst(3);

deque.offerLast(4);

// Removendo os elementos do deque

System.out.println(deque.pollFirst()); // Output: 3

System.out.println(deque.pollLast()); // Output: 4

}

}

**Output:**

3

4

#### A Interface Set

A Interface Set em Java está presente no pacote java.util, estendendo a interface Collection. Representa o conjunto não ordenado de elementos que não nos permite armazenar os itens duplicados. Podemos armazenar no máximo um valor nulo no Set. Set é implementado por HashSet, LinkedHashSet e TreeSet.

Set pode ser instanciado como:

Set<tipo de dado> s1 = new HashSet<tipo de dado>();

Set<tipo de dado> s2 = new LinkedHashSet<tipo de dado>();

Set<tipo de dado> s3 = new TreeSet<tipo de dado>();

##### A Classe AbstractSet

Essa classe fornece uma implementação esquelética da interface Set para minimizar o esforço necessário para implementar essa interface.

O processo de implementação da Classe conjunto estendendo esta classe é idêntico ao de implementar uma Coleção estendendo AbstractCollection, exceto que todos os métodos e construtores em subclasses desta classe devem obedecer às restrições adicionais impostas pela interface Set (por exemplo, o O método add não deve permitir a adição de várias instâncias da Classe objeto a um conjunto).

Observe que essa classe não substitui nenhuma das implementações da classe AbstractCollection. Ele simplesmente adiciona implementações para equals e hashCode.

###### A Interface SortedSet

SortedSet é a alternativa à Interface Set que fornece uma ordenação total de seus elementos. Os elementos do SortedSet são organizados na ordem crescente (ascendente). O SortedSet fornece os métodos adicionais que inibem a ordem natural dos elementos.

A Interface NavigableSet

NavigableSet representa um conjunto navegável no Java Collection Framework. A interface NavigableSet é herdada da interface SortedSet. Ele se comporta como um SortedSet com a exceção de que temos métodos de navegação disponíveis além dos mecanismos de classificação do SortedSet.

Por exemplo, a interface NavigableSet pode navegar no conjunto na ordem inversa em comparação com a ordem definida em SortedSet. Um NavigableSet pode ser acessado e percorrido em ordem crescente ou decrescente. As classes que implementam essa interface são TreeSet e ConcurrentSkipListSet

Exemplo:

public class NavigableSetDemo

{

public static void main(String[] args)

{

NavigableSet<Integer> ns = new TreeSet<>();

ns.add(0);

ns.add(1);

ns.add(2);

ns.add(3);

ns.add(4);

ns.add(5);

ns.add(6);

// Adquirindo a visão reversa do navigable set

NavigableSet<Integer> reverseNs = ns.descendingSet();

// Imprimindo a visão normal e invertida

System.out.println("Ordem normal: " + ns);

System.out.println("Ordem reversa: " + reverseNs);

NavigableSet<Integer> threeOrMore = ns.tailSet(3, true);

System.out.println("3 ou mais: " + threeOrMore);

System.out.println("menor(3): " + ns.lower(3));

System.out.println("chão(3): " + ns.floor(3));

System.out.println("maior(3): " + ns.higher(3));

System.out.println("teto(3): " + ns.ceiling(3));

System.out.println("pollFirst(): " + ns.pollFirst());

System.out.println("Navigable Set: " + ns);

System.out.println("pollLast(): " + ns.pollLast());

System.out.println("Navigable Set: " + ns);

System.out.println("pollFirst(): " + ns.pollFirst());

System.out.println("Navigable Set: " + ns);

System.out.println("pollFirst(): " + ns.pollFirst());

System.out.println("Navigable Set: " + ns);

System.out.println("pollFirst(): " + ns.pollFirst());

System.out.println("Navigable Set: " + ns);

System.out.println("pollFirst(): " + ns.pollFirst());

System.out.println("pollLast(): " + ns.pollLast());

}

}

**Output:**

Ordem normal: [0, 1, 2, 3, 4, 5, 6]

Ordem reversa: [6, 5, 4, 3, 2, 1, 0]

3 ou mais: [3, 4, 5, 6]

lower(3): 2

chão(3): 3

maior(3): 4

teto(3): 3

pollFirst(): 0

Navigable Set: [1, 2, 3, 4, 5, 6]

pollLast(): 6

Navigable Set: [1, 2, 3, 4, 5]

pollFirst(): 1

Navigable Set: [2, 3, 4, 5]

pollFirst(): 2

Navigable Set: [3, 4, 5]

pollFirst(): 3

Navigable Set: [4, 5]

pollFirst(): 4

pollLast(): 5

TreeSet

A classe Java TreeSet implementa a interface Set que usa uma árvore para armazenamento. Como o HashSet, o TreeSet também contém elementos exclusivos. No entanto, o tempo de acesso e recuperação do TreeSet é bastante rápido. Os elementos em TreeSet são armazenados em ordem crescente.

Exemplo:

class TestJavaCollection9{

static void main(String args[]){

TreeSet<String> set =new TreeSet<String>();

set.add("Ravi");

set.add("José");

set.add("Ravi");

set.add("Rose");

Iterator<String> itr=set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Output:**

José

Ravi

Rose

ConcurrentSkipListSet

A classe ConcurrentSkipListSet em Java faz parte do Java Collection Framework e implementa a interface Collection e a classe AbstractSet. Ele fornece uma versão escalável e simultânea do NavigableSet em Java. A implementação de ConcurrentSkipListSet é baseada em ConcurrentSkipListMap. Os elementos em ConcurrentSkipListSet são classificados por padrão em sua ordem natural ou por um Comparator fornecido no momento da criação do conjunto, dependendo de qual construtor é usado.

Uma vez que implementa SortedSet<E> e NavigableSet<E>, é semelhante a TreeSet com um recurso adicional de ser concorrente. Como é thread-safe, pode ser usado por vários threads simultaneamente, enquanto TreeSet não é thread-safe.

Exemplo:

class ConcurrentSkipListSetLastExample1 {

public static void main(String[] args)

{

// inicializando set utilizando

// ConcurrentSkipListSet()

ConcurrentSkipListSet<Integer> set

= new ConcurrentSkipListSet<Integer>();

// Adicionando elementos a este set

set.add(78);

set.add(64);

set.add(12);

set.add(45);

set.add(8);

// Imprimindo o ConcurrentSkipListSet

System.out.println("ConcurrentSkipListSet: " + set);

// Initializing the set using

// ConcurrentSkipListSet(Collection)

ConcurrentSkipListSet<Integer> set1

= new ConcurrentSkipListSet<Integer>(set);

// Imprimindo o ConcurrentSkipListSet1

System.out.println("ConcurrentSkipListSet1: "

+ set1);

// inicializando set utilizando

// ConcurrentSkipListSet()

ConcurrentSkipListSet<String> set2

= new ConcurrentSkipListSet<>();

// Adicionando elementos a este set

set2.add("Maçã");

set2.add("Limão");

set2.add("Banana");

set2.add("Maçã");

// criando um iterador

Iterator<String> itr = set2.iterator();

System.out.print("Set de Frutas: ");

while (itr.hasNext()) {

System.out.print(itr.next() + " ");

}

}

}

**Output:**

ConcurrentSkipListSet: [8, 12, 45, 64, 78]

ConcurrentSkipListSet1: [8, 12, 45, 64, 78]

Set de Frutas: Maçã, Banana, Limão

###### HashSet

A classe HashSet implementa a Interface Set. Representa a coleção que usa uma HashTable para armazenamento. Hashing é usado para armazenar os elementos no HashSet. Ele contém itens exclusivos.

Exemplo:

class TestJavaCollection7{

static void main(String args[]){

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

set.add("Ravi");

set.add("José");

set.add("Ravi");

set.add("Rose");

Iterator<String> itr = set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Output:**

José

Ravi

Rose

LinkedHashSet

A classe LinkedHashSet representa a implementação LinkedList na Interface Set. Ele estende a classe HashSet e implementa a interface Set. Como o HashSet, também contém elementos exclusivos. Mantém a ordem de inserção e permite elementos nulos.

class TestJavaCollection8{

static void main(String args[]){

LinkedHashSet<String> set = new LinkedHashSet<String>();

set.add("Ravi");

set.add("José");

set.add("Ravi");

set.add("Rose");

Iterator<String> itr = set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Output:**

José

Ravi

Rose

###### CopyOnWriteArraySet

CopyOnWriteArraySet é um membro do Java Collections Framework. É um Set que usa um [CopyOnWriteArrayList](#_A_Classe_CopyOnWriteArrayList) interno para todas as suas operações. Foi introduzido no JDK 1.5, podemos dizer que é uma versão thread-safe do Set.

Ele compartilha algumas propriedades de Set e também possui suas próprias propriedades listadas:

* A implementação interna de CopyOnWriteArraySet é apenas CopyOnWriteArrayList.
* Múltiplos Threads são capazes de executar operações de atualização simultaneamente, mas para cada operação de atualização, uma cópia clonada separada é criada. Como para cada atualização, uma nova cópia clonada será criada, o que é caro. Portanto, se várias operações de atualização forem necessárias, não é recomendável usar CopyOnWriteArraySet.
* Enquanto um thread itera o Set, outros threads podem realizar a atualização, aqui não teremos nenhuma exceção de tempo de execução como ConcurrentModificationException.
* Um Iterator da classe CopyOnWriteArraySet pode executar somente leitura e não deve executar a exclusão, caso contrário, obteremos a exceção de tempo de execução UnsupportedOperationException.
* Use CopyOnWriteArraySet em aplicativos nos quais os tamanhos dos conjuntos geralmente permanecem pequenos, as operações somente leitura superam amplamente as operações mutativas e você precisa evitar a interferência entre os threads durante a travessia.
* CopyOnWriteArraySet ajuda a minimizar as etapas de sincronização controladas pelo programador e a mover o controle para APIs incorporadas e bem testadas.

Exemplo:

class ConcurrentDemo extends Thread {

static CopyOnWriteArraySet l

= new CopyOnWriteArraySet();

// Método 1

public void run()

{

// A thread da criança ira tentar adicionar

// um novo elementos ao objeto Set

l.add("D");

}

// Método 2

public static void main(String[] args)

{

// Adicionando elementos

// utilizando o método add()

l.add("A");

l.add("B");

l.add("C");

// É criado a thread da criança

// que irá modificar o

// CopyOnWriteArraySet l.

ConcurrentDemo t = new ConcurrentDemo();

// Rodando a thread da criança

// utilizando o método start()

t.start();

// Esperando a thread

// adicionar o elemento

// Bloco try para checar por exceções

try {

Thread.sleep(2000);

}

// Bloco catch para lidar com as exceções

catch (InterruptedException e) {

System.out.println("a criança irá adicionar um elemento");

}

System.out.println(l);

// Agora é iterado pelo

// CopyOnWriteArraySet e não

// será retornado uma exceção

Iterator itr = l.iterator();

while (itr.hasNext()) {

String s = (String)itr.next();

System.out.println(s);

if (s.equals("C")) {

// Aqui irá ser retornado uma exceção

// RuntimeException

itr.remove();

}

}

}

}

**Output:**

[A, B, C, D]

A

B

C

Exception in thread "main" java.lang.UnsupportedOperationException

at java.base/java.util.concurrent.CopyOnWriteArrayList$COWIterator.remove(CopyOnWriteArrayList.java:1124)

at Ex/Ex.ConcurrentDemo.main(testesCollection.java:343)

at Ex/Ex.testesCollection.main(testesCollection.java:353)

###### EnumSet

Enumerações ou popularmente conhecidas como enum servem para representar um grupo de constantes nomeadas em uma linguagem de programação. Por exemplo, os 4 naipes em um baralho de cartas podem ser 4 enumeradores chamados Paus, Ouros, Copas e Espadas, pertencentes a um tipo enumerado chamado Naipe.

O EnumSet é uma das implementações especializadas da interface Set para uso com o tipo de enumeração. Alguns recursos importantes do EnumSet são os seguintes:

* Ele estende a classe AbstractSet e implementa Set Interface em Java.
* A classe EnumSet é membro do Java Collections Framework e não é sincronizada.
* É uma implementação de conjunto de alto desempenho, muito mais rápida que HashSet.
* Todos os elementos em um EnumSet devem vir da Classe único tipo de enumeração que é especificado quando o conjunto é criado explicitamente ou implicitamente.
* Ele não permite objetos nulos e lança NullPointerException se o fizermos.
* Ele usa um iterador à prova de falhas, portanto, não lançará ConcurrentModificationException se a coleção for modificada durante a iteração.

Exemplo:

enum DiasUteis {SEGUNDA, TERÇA, QUARTA, QUINTA, SEXTA};

class EnumSetExemplo {

public static void main(String[] args) {

// Creating a set

EnumSet<DiasUteis> set1, set2, set3, set4;

// Adding elements

set1 = EnumSet.of(DiasUteis.SEGUNDA,

DiasUteis.TERÇA, DiasUteis.QUARTA, DiasUteis.QUINTA);

set2 = EnumSet.complementOf(set1);

set3 = EnumSet.allOf(DiasUteis.class);

set4 = EnumSet.range(DiasUteis.SEGUNDA, DiasUteis.QUARTA);

// Printing corresponding elements in Sets

System.out.println("Set 1: " + set1);

System.out.println("Set 2: " + set2);

System.out.println("Set 3: " + set3);

System.out.println("Set 4: " + set4);

}

}

**Output:**

Set 1: [SEGUNDA, TERÇA, QUARTA, QUINTA]

Set 2: [SEXTA]

Set 3: [SEGUNDA, TERÇA, QUARTA, QUINTA, SEXTA]

Set 4: [SEGUNDA, TERÇA, QUARTA]

## A Interface Map

A interface Map em Java é uma das estruturas de dados que fazem parte do Collection Framework, que é um conjunto de interfaces e classes para representar e manipular coleções de objetos. A interface Map é uma das principais interfaces do Collection Framework e é usada para representar um mapeamento de chave-valor. Mesmo sendo parte do Collection Framework, ela não pode ser totalmente considerada uma coleção por conta de algumas de suas propriedades

Enquanto as coleções como List, Set e Queue são usadas para armazenar grupos de objetos, a interface Map é usada para armazenar pares de chave-valor. Ou seja, cada elemento em um mapa é um par de objetos, onde um é a chave e o outro é o valor correspondente. As chaves em um Map são únicas, enquanto os valores podem ser duplicados.

Uma das principais vantagens de usar a interface Map é a capacidade de recuperar um valor armazenado rapidamente usando a chave correspondente. Em vez de percorrer todos os elementos em uma coleção para encontrar um objeto específico, podemos usar a chave correspondente para recuperar o valor diretamente.

### A Interface Map.Entry

A interface Map.Entry é uma subinterface da interface Map em Java, usada para representar um par chave-valor em um mapa. Ela encapsula uma entrada em um mapa e é retornada por vários métodos do mapa, como entrySet(), que retorna um conjunto de todas as entradas no mapa.

Exemplo Não Genérico (Estilo Antigo):

//Não Genérico

class MapExample1 {

static void main(String[] args) {

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

//Adicionado elementods ao mapa

map.put(1,"Remmi");

map.put(5,"Rahul");

map.put(2,"Nay");

map.put(6,"Sant");

//Atravessando Mapa

Set set = map.entrySet();//Convertendo para Set para que possamos percorrer

Iterator itr=set.iterator();

while(itr.hasNext()){

//Convertendo para Map.Entry para que possamos obter chave e valor separadamente

Map.Entry entry=(Map.Entry)itr.next();

System.out.println(entry.getKey()+" "+entry.getValue());

}

}

}

**Output:**

Remmi

Rahul

Nay

Sant

Exemplo Genérico (Novo Estilo):

class MapExample2{

static void main(String args[]){

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

map.put(100,"Yna");

map.put(101,"Spec");

map.put(102,"Rahul");

//Elements can traverse in any order

for(Map.Entry m:map.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

**Output:**

100 Yna

101 Spec

102 Rahul

Exemplo comparingByKey():

class MapExample3{

static void main(String args[]){

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

map.put(100,"Any");

map.put(101,"Joseph");

map.put(102,"Rafael");

//Retorna uma visualização Set dos mapeamentos contidos neste mapa

map.entrySet()

//Retorna um Stream sequencial com esta coleção como fonte

.stream()

//Classificado de acordo com o Comparador fornecido

.sorted(Map.Entry.comparingByKey())

//Executa uma ação para cada elemento deste fluxo

.forEach(System.out::println);

}

}

**Output:**

100=Any

101=Joseph

102=Rafael

Exemplo comparingByKey(): (por ordem decrescente)

class MapExample4{

static void main(String args[]){

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

map.put(100,"Any");

map.put(101,"Joseph");

map.put(102,"Rafael");

//Retorna uma visualização Set dos mapeamentos contidos neste mapa

map.entrySet()

//Retorna um Stream sequencial com esta coleção como fonte

.stream()

//Classificado de acordo com o Comparador fornecido

.sorted(Map.Entry.comparingByKey(Comparator.reverseOrder()))

//Executa uma ação para cada elemento deste fluxo

.forEach(System.out::println);

}

}

**Output:**

102=Rafael

101=Joseph

100=Any

Exemplo comparingByValue():

class MapExample5{

static void main(String args[]){

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

map.put(100,"Any");

map.put(101,"Joseph");

map.put(102,"Rafael");

//Retorna uma visualização Set dos mapeamentos contidos neste mapa

map.entrySet()

//Retorna um Stream sequencial com esta coleção como fonte

.stream()

//Classificado de acordo com o Comparador fornecido

.sorted(Map.Entry.comparingByValue())

//Executa uma ação para cada elemento deste fluxo

.forEach(System.out::println);

}

}

**Output:**

100=Any

101=Joseph

102=Rafael

Exemplo comparingByValue(): (por ordem decrescente)

class MapExample5{

static void main(String args[]){

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

map.put(100,"Any");

map.put(101,"Joseph");

map.put(102,"Rafael");

//Retorna uma visualização Set dos mapeamentos contidos neste mapa

map.entrySet()

//Retorna um Stream sequencial com esta coleção como fonte

.stream()

//Classificado de acordo com o Comparador fornecido

.sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))

//Executa uma ação para cada elemento deste fluxo

.forEach(System.out::println);

}

}

**Output:**

102=Rafael

101=Joseph

100=Any

### A Classe AbstractMap

A classe AbstractMap faz parte do Java Collection Framework. Ele implementa diretamente a interface Map para fornecer uma estrutura a ela, ao fazer isso, facilita as implementações posteriores. Como o nome sugere, AbstractMap é uma classe abstrata por definição, portanto não pode ser usada para criar objetos, Pórem as classes concretas que herdam do AbstractMap podem ser utilizadas para criar objetos.

#### A Interface ConcurrentMap

A interface ConcurrentMap é uma subinterface da interface Map em Java que foi introduzida a partir da versão 1.5 do Java. Ela define um mapa que pode ser acessado e atualizado simultaneamente por várias threads sem a necessidade de bloqueio.

A principal diferença entre a ConcurrentMap e a interface Map padrão é que a ConcurrentMap fornece operações seguras para uso em ambientes concorrentes. As operações de leitura (como get) podem ser executadas simultaneamente por várias threads sem problemas, mas as operações de escrita (como put) exigem mais atenção, pois duas ou mais threads podem tentar escrever no mapa ao mesmo tempo.

A interface ConcurrentMap fornece métodos adicionais para lidar com esses problemas de concorrência, como putIfAbsent(K key, V value), que adiciona um par de chave-valor ao mapa se a chave ainda não existir, ou replace(K key, V oldValue, V newValue), que substitui um valor existente na chave especificada somente se o valor antigo ainda for o mesmo.

Além disso, a interface ConcurrentMap fornece métodos para realizar operações atômicas, como putIfAbsent(K key, V value), que adiciona um par de chave-valor ao mapa somente se a chave ainda não existir. Esses métodos são úteis quando várias threads precisam atualizar o mapa ao mesmo tempo.

##### ConcurrentHashMap

O ConcurrentHashMap é uma implementação de mapa que é projetada para ser usada em ambientes concorrentes, onde vários threads podem acessar e modificar o mapa simultaneamente. Ele é uma variante thread-safe da classe HashMap padrão em Java.

A principal diferença entre o ConcurrentHashMap e o HashMap é que o primeiro é projetado para suportar operações concorrentes, enquanto o último não é. O ConcurrentHashMap é projetado para lidar com situações em que várias threads podem tentar adicionar, atualizar ou remover elementos do mapa ao mesmo tempo, sem que ocorra condições de corrida (race conditions), perda de dados ou corrupção do mapa.

O ConcurrentHashMap usa uma estrutura interna de dados chamada de "tabela hash escalonável" (scalable hash table), que é dividida em segmentos independentes. Cada segmento é basicamente um mapa hash separado e é gerenciado de forma independente, o que permite que várias threads possam acessar diferentes segmentos do mapa ao mesmo tempo.

O ConcurrentHashMap fornece um conjunto de operações que são seguras para uso concorrente, como put(), get(), remove() e outras. Além disso, ele também suporta operações atômicas como putIfAbsent(), replace() e outras.

Aqui está um exemplo simples de como usar o ConcurrentHashMap em Java:

Exemplo:

public class ExemploConcurrentHashMap {

public static void main(String[] args) {

// Criando um novo ConcurrentHashMap

ConcurrentHashMap<String, Integer> mapa = new ConcurrentHashMap<>();

// Adicionando elementos ao mapa

mapa.put("Alice", 25);

mapa.put("Bob", 30);

mapa.put("Carol", 35);

mapa.put("Dave", 40);

// Imprimindo o mapa

System.out.println("Mapa: " + mapa);

// Obtendo o valor associado a uma chave

int idadeBob = mapa.get("Bob");

System.out.println("Idade do Bob: " + idadeBob);

// Removendo um elemento do mapa

mapa.remove("Carol");

// Imprimindo o mapa atualizado

System.out.println("Mapa atualizado: " + mapa);

}

}

##### A Interface ConcurrentNavigableMap

A Interface ConcurrentNavigableMap é uma subinterface da Interface ConcurrentMap que estende a Interface NavigableMap e fornece um mapa navegável e concorrente, ou seja, que pode ser acessado e modificado por várias threads simultaneamente. Essa interface permite que os usuários acessem, naveguem e manipulem as entradas do mapa em ordem de chave, com suporte para operações atômicas em conjunto com iterações seguras para várias threads.

A Interface ConcurrentNavigableMap fornece muitos métodos adicionais em relação à Interface NavigableMap. Alguns desses métodos incluem:

ConcurrentNavigableMap<K, V> subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive): retorna uma visão do mapa que contém as entradas com chaves entre fromKey e toKey, incluindo ou excluindo as extremidades, dependendo dos argumentos booleanos fromInclusive e toInclusive.

ConcurrentNavigableMap<K, V> headMap(K toKey, boolean inclusive): retorna uma visão do mapa que contém as entradas com chaves menores do que toKey, incluindo ou excluindo toKey, dependendo do argumento booleano inclusive.

ConcurrentNavigableMap<K, V> tailMap(K fromKey, boolean inclusive): retorna uma visão do mapa que contém as entradas com chaves maiores do que ou iguais a fromKey, incluindo ou excluindo fromKey, dependendo do argumento booleano inclusive.

ConcurrentNavigableMap<K, V> descendingMap(): retorna uma visão reversa do mapa.

Além disso, a Interface ConcurrentNavigableMap também herda todos os métodos da Interface ConcurrentMap e da Interface NavigableMap.

Algumas implementações populares da Interface ConcurrentNavigableMap incluem ConcurrentSkipListMap, que implementa um mapa navegável e concorrente usando uma lista vinculada, e ConcurrentSkipListSet, que implementa um conjunto navegável e concorrente usando a mesma estrutura de dados do ConcurrentSkipListMap.

###### ConcurrentSkipListMap

O ConcurrentSkipListMap é uma estrutura de dados em forma de mapa implementada na linguagem de programação Java que permite armazenar pares de chave-valor. Ela é chamada de "Skip List" por ser baseada em uma estrutura de dados chamada "Skip List", que permite uma busca eficiente em uma lista ordenada.

A principal diferença entre o ConcurrentSkipListMap e outros mapas é que ele é projetado para ser usado em ambientes multi-threaded, ou seja, em que várias threads podem acessá-lo simultaneamente. Isso é possível devido à sua implementação, que utiliza uma série de listas ligadas paralelas, cada uma com um número diferente de elementos, de forma que a lista mais curta é um subconjunto da lista mais longa.

A implementação do ConcurrentSkipListMap é baseada em operações atômicas, o que garante a consistência dos dados mesmo em ambientes de concorrência. Além disso, ele fornece uma série de operações de acesso e modificação dos dados, como put(), get(), remove() e outras.

Por ser uma estrutura de dados eficiente para a realização de operações de leitura e gravação simultâneas em um ambiente de concorrência, o ConcurrentSkipListMap é frequentemente usado em sistemas que lidam com grandes volumes de dados em tempo real, como aplicações financeiras, sistemas de gerenciamento de dados e aplicações de jogos online.

Exemplo:

public class ExemploConcurrentSkipListMap {

public static void main(String[] args) {

// Criando um novo ConcurrentSkipListMap

ConcurrentSkipListMap<String, Integer> mapa = new ConcurrentSkipListMap<>();

// Adicionando elementos ao mapa

mapa.put("Alice", 25);

mapa.put("Bob", 30);

mapa.put("Carol", 35);

mapa.put("Dave", 40);

// Imprimindo o mapa

System.out.println("Mapa: " + mapa);

// Obtendo o valor associado a uma chave

int idadeBob = mapa.get("Bob");

System.out.println("Idade do Bob: " + idadeBob);

// Removendo um elemento do mapa

mapa.remove("Carol");

// Imprimindo o mapa atualizado

System.out.println("Mapa atualizado: " + mapa);

}

}

#### A Interface SortedMap

A Interface SortedMap é uma subinterface da Interface Map em Java que estende a funcionalidade da Interface Map para fornecer uma coleção ordenada de pares de chave-valor com base na ordem das chaves. A interface SortedMap exige que suas chaves sejam ordenáveis e implementem a interface Comparable ou que um Comparator seja fornecido para ordenar as chaves.

A principal diferença entre a Interface SortedMap e a Interface Map padrão é que a SortedMap mantém as chaves em ordem natural ou em uma ordem definida por um Comparator. Isso permite que os usuários recuperem submapas com base em um intervalo de chaves ou obtenham as chaves em ordem crescente ou decrescente.

A Interface SortedMap define métodos adicionais que permitem que os usuários obtenham submapas com base em um intervalo de chaves, como subMap(K fromKey, K toKey) e headMap(K toKey). A interface também define métodos para obter o comparador usado para classificar as chaves (comparator()) e para obter as chaves em ordem crescente ou decrescente (keySet(), values(), entrySet()).

Algumas implementações populares da Interface SortedMap incluem TreeMap, que implementa um mapa ordenado usando uma estrutura de árvore balanceada e ConcurrentSkipListMap, que implementa um mapa ordenado usando uma lista vinculada.

##### A Interface NavigableMap

A Interface NavigableMap é uma subinterface da Interface SortedMap em Java que fornece uma coleção de pares de chave-valor ordenados e navegáveis com base na ordem natural das chaves ou em uma ordem definida por um Comparator. Essa interface permite que os usuários pesquisem, naveguem e manipulem as entradas do mapa com base em sua posição na ordem das chaves.

A Interface NavigableMap fornece muitos métodos adicionais em relação à Interface SortedMap. Alguns desses métodos incluem:

lowerEntry(K key) e lowerKey(K key): retorna o maior par de chave-valor na ordem das chaves que é menor do que a chave especificada.

floorEntry(K key) e floorKey(K key): retorna o maior par de chave-valor na ordem das chaves que é menor ou igual à chave especificada.

ceilingEntry(K key) e ceilingKey(K key): retorna o menor par de chave-valor na ordem das chaves que é maior ou igual à chave especificada.

higherEntry(K key) e higherKey(K key): retorna o menor par de chave-valor na ordem das chaves que é maior do que a chave especificada.

firstEntry() e firstKey(): retorna o menor par de chave-valor na ordem das chaves.

lastEntry() e lastKey(): retorna o maior par de chave-valor na ordem das chaves.

pollFirstEntry() e pollLastEntry(): remove e retorna o menor ou o maior par de chave-valor na ordem das chaves.

Além disso, a Interface NavigableMap também herda todos os métodos da Interface SortedMap, como subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive), que retorna um submapa que inclui todas as chaves entre fromKey e toKey, incluindo ou excluindo as extremidades, dependendo dos argumentos booleanos fromInclusive e toInclusive.

Algumas implementações populares da Interface NavigableMap incluem TreeMap, que implementa um mapa ordenado usando uma estrutura de árvore balanceada, e ConcurrentSkipListMap, que implementa um mapa ordenado usando uma lista vinculada.

###### TreeMap

O TreeMap é uma classe que implementa a interface SortedMap e é uma das implementações de mapas. Ela é uma estrutura de dados de árvore balanceada em que os elementos são armazenados em uma ordem natural ou por meio de um comparador definido pelo usuário.

Ao contrário do HashMap, em que a ordem dos elementos não é garantida, o TreeMap armazena os elementos em uma estrutura ordenada, o que permite que eles sejam recuperados de acordo com uma ordem definida. Essa ordem é determinada pelo valor da chave de cada elemento, e não pelo valor do elemento em si.

A principal vantagem do TreeMap é que ele permite uma busca eficiente de elementos, com um tempo de pesquisa em ordem logarítmica (O(log n)). Isso ocorre porque a estrutura de dados é organizada em uma árvore, em que cada nó tem no máximo dois filhos, e todos os nós da subárvore esquerda têm valores menores que o nó raiz, enquanto todos os nós da subárvore direita têm valores maiores.

Além disso, o TreeMap fornece métodos para obter submapas, como headMap, tailMap e subMap, que permitem que você selecione um conjunto de elementos com base em uma chave inicial e final.

No entanto, é importante notar que o TreeMap pode ser menos eficiente em termos de memória e desempenho do que outras implementações de mapas, como o HashMap. Além disso, ele pode ser mais complicado de usar em certas situações, pois requer que as chaves sejam comparáveis ou que um comparador seja fornecido.

Exemplo:

public class TreeMapExample {

public static void main(String[] args) {

// Cria um TreeMap vazio com a ordem natural das chaves

TreeMap<String, Integer> treeMap = new TreeMap<>();

// Adiciona elementos ao TreeMap

treeMap.put("Ana", 25);

treeMap.put("Bruno", 33);

treeMap.put("Clara", 19);

treeMap.put("Daniel", 42);

// Recupera e imprime os elementos do TreeMap

System.out.println(treeMap);

// Recupera um submapa do TreeMap

System.out.println(treeMap.subMap("B", "D"));

// Remove um elemento do TreeMap

treeMap.remove("Clara");

// Verifica se o TreeMap contém uma chave específica

System.out.println(treeMap.containsKey("Daniel"));

// Recupera o valor associado a uma chave específica

int idadeDaniel = treeMap.get("Daniel");

System.out.println("A idade de Daniel é " + idadeDaniel);

}

}

#### HashMap

A classe Java HashMap implementa a interface Map que nos permite armazenar pares chave e valor, onde as chaves devem ser únicas. Se você tentar inserir a chave duplicada, ela substituirá o elemento da chave correspondente. É fácil executar operações usando o índice de chave como atualização, exclusão, etc. A classe HashMap é também encontrada no pacote java.util.

O HashMap em Java é como a classe Hashtable, porém não é sincronizada. Ele também nos permite armazenar os elementos nulos, mas pode haver apenas uma chave nula. Desde o Java 5, ele é indicado como HashMap<K,V>, onde K significa chave e V, valor. Ele também herda a classe AbstractMap e implementa a interface Map.

Exemplo:

class HashMapExample1{

static void main(String args[]){

//Criando o HashMap

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

map.put(1,"Manga"); //Colocando elementos no mapa

map.put(2,"Maça");

map.put(3,"Uva");

map.put(4,"Jabuticaba");

System.out.println("Iterando...");

for(Map.Entry m : map.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

**Output:**

Iterando...

1 Manga

2 Maça

3 Uva

4 Jabuticaba

Neste exemplo, estamos armazenandoo tipo de dado Integer como a chave e o tipo de dado String como o valor, então estamos usando HashMap<Integer,String> como o tipo. O método put() insere os elementos no mapa.

Para obter os elementos chave e valor, devemos chamar os métodos getKey() e getValue(). A interface Map.Entry contém os métodos getKey() e getValue(). Porém, devemos chamar o método entrySet() da interface Map para obter a instância do Map.Entry.

Você não pode armazenar chaves duplicadas no HashMap. No entanto, se você tentar armazenar a chave duplicada com outro valor, ela substituirá o valor.

Exemplo:

class HashMapExample2{

static void main(String args[]){

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

map.put(1,"Manga");

map.put(2,"Maça");

map.put(3,"Uva");

map.put(1,"Jabuticaba"); //tentando duplicar uma key

for(Map.Entry m : map.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

**Output:**

1 Jabuticaba

2 Maça

3 Uva

##### HashMap x HashSet

HashSet é uma implementação de Set em que cada elemento é único. Não permite elementos duplicados e não tem nenhuma ordem particular dos elementos. Quando você adiciona um elemento a um HashSet, o elemento é verificado se já existe no conjunto. Se não existir, ele é adicionado. Se já existir, o elemento não é adicionado novamente. Portanto, HashSet é uma boa escolha quando você precisa armazenar uma coleção de objetos exclusivos, e não se preocupa com a ordem dos elementos.

Por outro lado, HashMap é uma implementação de Map que armazena os elementos como pares chave-valor. Cada elemento em um HashMap é uma associação de uma chave exclusiva com um valor correspondente. Você pode adicionar um par chave-valor a um HashMap e obter o valor correspondente usando a chave. Ele permite chaves e valores duplicados, mas a chave deve ser exclusiva. E também não garante nenhuma ordem particular dos elementos.

Portanto, a principal diferença entre HashSet e HashMap é que HashSet armazena apenas elementos únicos e não tem nenhuma ordem particular dos elementos, enquanto HashMap armazena pares chave-valor, permitindo valores duplicados, e também não garante nenhuma ordem particular dos elementos.

##### LinkedHashMap

LinkedHashMap é uma implementação do Map em que a ordem dos elementos é mantida, o que significa que os elementos são iterados na ordem em que foram inseridos ou na ordem de sua última atualização, dependendo do modo de ordenação definido. Isso é possível porque ele é uma extensão da classe HashMap, mas mantém uma lista duplamente vinculada dos elementos para manter a ordem de inserção.

Assim como o HashMap, o LinkedHashMap permite pares chave-valor e fornece as mesmas operações básicas, como put(), get(), remove(), containsKey(), entre outras. Mas a diferença é que o ele adiciona mais duas operações específicas para ordenação:

* public void put(int index, K key, V value): Adiciona um novo par chave-valor no índice especificado.
* public void setOrder(boolean accessOrder): Define a ordem de iteração dos elementos. Se accessOrder for true, a ordem será baseada na última vez em que o elemento foi acessado (útil para cache e algoritmos de remoção de itens mais antigos). Caso contrário, a ordem será baseada na ordem de inserção.

Resumindo ele é uma boa opção para situações em que é necessário manter a ordem de inserção dos elementos ou para implementações de cache, em que os elementos menos acessados podem ser removidos primeiro. No entanto, é importante notar que a manutenção da ordem de inserção requer um pouco mais de overhead de memória e processamento em comparação com HashMap, especialmente quando a coleção é grande.

Exemplo:

class LinkedHashMap1{

static void main(String args[]){

LinkedHashMap<Integer,String> lhm = new LinkedHashMap <Integer,String>();

lhm.put(100,"Rahul");

lhm.put(101,"Nay");

lhm.put(102,"Sant");

for(Map.Entry m:lhm.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

**Output:**

100 Rahul

101 Nay

102 Sant

Exemplo Par de Chave e Valor:

class LinkedHashMap2{

static void main(String args[]){

LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>();

map.put(100,"Rahul");

map.put(101,"Nay");

map.put(102,"Sant");

//Buscando a chave

System.out.println("Chaves: "+map.keySet());

//Buscando o valor

System.out.println("Valores: "+map.values());

//Buscando os pares das chaves e valores

System.out.println("Pares de Chaves e Valores: "+map.entrySet());

}

}

#### WeakHashMap

WeakHashMap é uma implementação da interface Map em Java que usa referências fracas (weak references) para as chaves da coleção. Isso significa que as chaves em um WeakHashMap são armazenadas como referências fracas em vez de referências fortes, o que permite que elas sejam coletadas pelo coletor de lixo (garbage collector) quando não há mais nenhuma referência forte a elas.

Essa característica é útil quando se deseja associar valores a objetos que podem ser coletados pelo coletor de lixo, como objetos temporários, caches ou objetos usados em cache de objetos que podem ser reconstruídos. Quando as chaves são coletadas, o correspondente valor é automaticamente removido do mapa, o que evita vazamentos de memória e reduz o consumo de memória.

No entanto, é importante lembrar que o uso de WeakHashMap pode ter impacto negativo no desempenho do programa, uma vez que o coletor de lixo precisa ser executado com mais frequência para remover as chaves que não têm referências fortes. Portanto, é recomendado usar WeakHashMap apenas quando for necessário associar valores a objetos que podem ser coletados pelo coletor de lixo.

Exemplo:

public class Main {

public static void main(String[] args) {

Map<String, Integer> weakHashMap = new WeakHashMap<>();

String key1 = new String("chave1");

String key2 = new String("chave2");

String key3 = new String("chave3");

weakHashMap.put(key1, 10);

weakHashMap.put(key2, 20);

weakHashMap.put(key3, 30);

System.out.println(weakHashMap.get(key1)); // Saída: 10

// Remove a referência forte para a chave "key2"

key2 = null;

// Garbage collector é chamado explicitamente para liberar memória

System.gc();

System.out.println(weakHashMap.get(key1)); // Saída: 10

System.out.println(weakHashMap.get(key3)); // Saída: 30

}

}

#### IndentityHashMap

O IdentityHashMap é uma implementação da interface Map que usa a identidade dos objetos como chaves na coleção, em vez de comparar as chaves usando o método equals() e hashCode() como é feito em outras implementações de Map.

Na prática, isso significa que, em um IdentityHashMap, duas chaves são consideradas iguais apenas se elas forem o mesmo objeto (têm a mesma referência de memória), em vez de terem valores iguais ou serem iguais de acordo com a definição de igualdade da classe da chave.

Essa característica é útil em situações em que as chaves são objetos mutáveis que podem ter seu valor alterado durante a execução do programa, mas a identidade do objeto deve ser mantida como chave.

No entanto, é importante observar que o seu uso pode ser confuso se as chaves não forem objetos imutáveis ou se a identidade dos objetos não for claramente definida. Além disso, seu desempenho pode ser pior do que outras implementações de Map devido à comparação de identidade em vez da comparação de valores. Portanto, deve-se utilizar IdentityHashMap apenas em situações específicas em que a comparação por identidade é necessária.

Exemplo:

public class Main {

public static void main(String[] args) {

IdentityHashMap<String, String> identityHashMap = new IdentityHashMap<>();

String key1 = "chave";

String key2 = new String("chave");

identityHashMap.put(key1, "valor1");

identityHashMap.put(key2, "valor2");

System.out.println(identityHashMap.get(key1)); // Saída: valor1

System.out.println(identityHashMap.get(key2)); // Saída: valor2

}

#### EnumMap

O EnumMap é uma implementação especializada da interface Map em Java que usa chaves do tipo enum. Ela é otimizada para chaves do tipo enum e garante que as chaves são sempre mantidas em ordem natural de enumeração.

Ao contrário de outras implementações de Map, EnumMap não permite chaves nulas e não é thread-safe. No entanto, ela oferece um bom desempenho em situações em que se precisa armazenar valores associados a chaves do tipo enum. Além disso, EnumMap possui algumas vantagens em relação a outras implementações de Map, como:

* Melhor desempenho: EnumMap é otimizada para chaves do tipo enum e é mais rápida do que outras implementações de Map quando se trabalha com chaves do tipo enum.
* Menor consumo de memória: EnumMap usa um array interno para armazenar os valores, o que significa que é mais eficiente em termos de memória do que outras implementações de Map.
* Chaves ordenadas: EnumMap garante que as chaves são mantidas em ordem natural de enumeração, o que pode ser útil em algumas situações.

Para criar um EnumMap, é necessário passar a classe enum como parâmetro no construtor. Por exemplo:

EnumMap<MyEnum, String> enumMap = new EnumMap<>(MyEnum.class);

Neste exemplo, MyEnum é a classe enum usada como chave no EnumMap. Uma vez criado o EnumMap, é possível adicionar valores associados a cada chave da seguinte forma:

enumMap.put(MyEnum.VALUE1, "Valor 1");

enumMap.put(MyEnum.VALUE2, "Valor 2");

O EnumMap garante que as chaves serão mantidas em ordem natural de enumeração (VALUE1, VALUE2, etc.). Além disso, o EnumMap é mais eficiente em termos de memória do que outras implementações de Map e oferece melhor desempenho quando se trabalha com chaves do tipo enum.

Exemplo:

enum MyEnum {

VALUE1,

VALUE2,

VALUE3

}

public class Main {

public static void main(String[] args) {

EnumMap<MyEnum, String> enumMap = new EnumMap<>(MyEnum.class);

enumMap.put(MyEnum.VALUE1, "Valor 1");

enumMap.put(MyEnum.VALUE2, "Valor 2");

enumMap.put(MyEnum.VALUE3, "Valor 3");

System.out.println(enumMap.get(MyEnum.VALUE1)); // Saída: Valor 1

System.out.println(enumMap.get(MyEnum.VALUE2)); // Saída: Valor 2

System.out.println(enumMap.get(MyEnum.VALUE3)); // Saída: Valor 3

}

}

### A Classe Dictionary

A classe Dictionary é uma classe abstrata em Java que representa uma coleção de pares de chave-valor. Ela foi introduzida nas primeiras versões do Java e foi a base para outras classes de mapa, como a Hashtable e a HashMap. Ela é uma classe abstrata e não possui métodos públicos para serem chamados diretamente. Em vez disso, as classes filhas devem implementar seus próprios métodos para fornecer funcionalidade específica.

A classe Dictionary define dois métodos abstratos que devem ser implementados por suas subclasses:

* public abstract int size(): retorna o número de entradas no dicionário.
* public abstract Object get(Object key): retorna o valor associado a uma chave no dicionário.

Além desses métodos, a classe Dictionary também oferece alguns métodos concretos (já implementados) que podem ser usados por suas subclasses. Alguns desses métodos são:

* public abstract Object put(Object key, Object value): adiciona uma entrada ao dicionário, associando uma chave a um valor. Se a chave já estiver presente, o valor antigo será substituído pelo novo valor e o valor antigo será retornado. Caso contrário, o valor nulo será retornado.
* public abstract Object remove(Object key): remove uma entrada do dicionário, dada uma chave. Retorna o valor associado à chave ou nulo, se a chave não estiver presente no dicionário.
* public abstract Enumeration elements(): retorna uma enumeração dos valores contidos no dicionário.
* public abstract Enumeration keys(): retorna uma enumeração das chaves contidas no dicionário.

A classe Dictionary é usada como uma classe base para a implementação de outros tipos de mapas em Java. Por exemplo, a classe Hashtable é uma subclasse de Dictionary e oferece a mesma funcionalidade, mas com melhorias na eficiência e na segurança de thread. No entanto, em aplicações modernas, é mais comum utilizar outras classes de mapa, como a HashMap e a TreeMap, que oferecem recursos mais avançados e uma maior flexibilidade.

Exemplo:

public class ExemploDictionary {

public static void main(String[] args) {

// Criando um novo dicionário

Dictionary<String, Integer> dict = new Hashtable<>();

// Adicionando entradas ao dicionário

dict.put("um", 1);

dict.put("dois", 2);

dict.put("três", 3);

// Imprimindo o tamanho do dicionário

System.out.println("Tamanho do dicionário: " + dict.size());

// Acessando um valor no dicionário através da chave

System.out.println("Valor associado à chave 'dois': " + dict.get("dois"));

// Removendo uma entrada do dicionário

dict.remove("três");

// Imprimindo o tamanho do dicionário novamente

System.out.println("Tamanho do dicionário após remover uma entrada: " + dict.size());

}

}

#### HashTable

A classe Hashtable é uma implementação da interface Map em Java que fornece um mapeamento de chave-valor sincronizado. Ela armazena os pares de chave-valor em uma tabela de hash e usa a função hash da chave para localizar o valor correspondente.

Algumas das principais características da classe Hashtable são:

* Ela é sincronizada, o que significa que vários threads podem acessá-la ao mesmo tempo sem causar problemas de concorrência. Isso torna a Hashtable uma boa escolha em ambientes com múltiplos threads concorrentes.
* Ela não permite chaves ou valores nulos. Se você tentar inserir uma chave ou valor nulo, a Hashtable lançará uma exceção NullPointerException.
* Ela é ordenada por hash, o que significa que não há garantia de ordem de iteração dos elementos.

A classe Hashtable oferece vários métodos para adicionar, remover e acessar elementos na tabela. Alguns dos métodos mais comuns são:

* public synchronized V put(K key, V value): adiciona um par chave-valor à tabela. Se a chave já existir na tabela, o valor antigo será substituído pelo novo valor e o valor antigo será retornado.
* public synchronized V get(Object key): retorna o valor associado a uma determinada chave na tabela.
* public synchronized V remove(Object key): remove o par chave-valor correspondente à chave especificada.
* public synchronized boolean containsKey(Object key): verifica se a tabela contém uma determinada chave.
* public synchronized boolean containsValue(Object value): verifica se a tabela contém um determinado valor.

A classe Hashtable é considerada obsoleta em comparação com outras implementações mais recentes de mapa em Java, como a HashMap e a ConcurrentHashMap. No entanto, ela ainda pode ser útil em ambientes onde a sincronização é necessária e a ordem de iteração dos elementos não é importante.

Exemplo:

public class ExemploHashtable {

public static void main(String[] args) {

// Criando uma nova Hashtable

Hashtable<String, Integer> hashtable = new Hashtable<>();

// Adicionando entradas à tabela hash

hashtable.put("um", 1);

hashtable.put("dois", 2);

hashtable.put("três", 3);

// Imprimindo o tamanho da tabela hash

System.out.println("Tamanho da tabela hash: " + hashtable.size());

// Acessando um valor na tabela hash usando a chave

System.out.println("Valor associado à chave 'dois': " + hashtable.get("dois"));

// Removendo uma entrada da tabela hash

hashtable.remove("três");

// Imprimindo o tamanho da tabela hash novamente

System.out.println("Tamanho da tabela hash após remover uma entrada: " + hashtable.size());

}

}

##### HashTable x HashMap

A principal diferença entre a classe Hashtable e a classe HashMap em Java é que a Hashtable é sincronizada e a HashMap não é. Isso significa que as operações em uma Hashtable são thread-safe, o que é importante em ambientes com múltiplos threads concorrentes. Por outro lado, a classe HashMap não é thread-safe, o que significa que deve ser usada com cuidado em ambientes com múltiplos threads.

Além disso, a classe Hashtable não permite chaves ou valores nulos, enquanto a classe HashMap permite. Se você tentar inserir uma chave ou valor nulo em uma Hashtable, ela lançará uma exceção NullPointerException. Já na classe HashMap, é possível inserir chaves e valores nulos sem problema.

A ordem de iteração dos elementos também é diferente nas duas classes. Na classe Hashtable, a ordem de iteração é a ordem em que os elementos foram inseridos. Na classe HashMap, a ordem de iteração não é especificada e pode mudar a cada vez que você itera sobre os elementos.

Por fim, vale lembrar que a classe Hashtable é considerada obsoleta em comparação com a classe HashMap. Isso ocorre porque a classe Hashtable tem um desempenho pior do que a classe HashMap em ambientes com apenas um thread. Se você precisa de sincronização em ambientes com múltiplos threads, é recomendado o uso da classe ConcurrentHashMap em vez da Hashtable.

# Catálogo de Métodos

## Métodos da Interface Collection

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | É utilizado para inserir um elemento em uma coleção |
| boolean | addAll(Collection<? extends E> c) | Ele é usado para inserir os elementos de coleção especificados na coleção de chamada. |
| void | clear() | Ele remove o número total de elementos da coleção. |
| boolean | contains(Object element) | É usado para pesquisar um elemento. |
| boolean | containsAll(Collection<?> c) | Ele é usado para pesquisar a coleção especificada na coleção. |
| boolean | equals(Object element) | Compara duas coleções. |
| int | hashCode() | Ele retorna o número do código hash da coleção. |
| boolean | isEmpty() | Ele verifica se a coleção está vazia. |
| Iterator<E> | iterator() | Retorna um iterador. |
| default Stream<E> | parallelStream() | Ele retorna um Stream possivelmente paralelo com a coleção como sua fonte. |
| boolean | remove(Object element) | É usado para excluir um elemento da coleção. |
| boolean | removeAll(Collection<?> c) | Ele é usado para excluir todos os elementos da coleção especificada da coleção de chamada. |
| default boolean | removeIf(Predicate<? super E> filter) | Ele é usado para excluir todos os elementos da coleção que satisfazem o predicado especificado. |
| boolean | retainAll(Collection<?> c) | Ele é usado para excluir todos os elementos da coleção invocada, exceto a coleção especificada. |
| int | size() | Ele retorna o número total de elementos na coleção. |
| default Spliterator<E> | spliterator() | Ele gera um Spliterator sobre os elementos especificados na coleção. |
| default Stream<E> | stream() | Ele retorna um Stream sequencial com a coleção como sua origem. |
| Object[] | toArray() | Converte uma coleção em uma array. |
| <T> T[] | toArray(T[] a) | Ele converte coleção em array. Aqui, o tipo de tempo de execução da array retornada é o da array especificada. |

## Métodos da Interface Iterador

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | hasNext() | Retorna verdadeiro se o iterador tiver mais elementos, caso contrário, retorna falso. |
| Object | next() | Ele retorna o elemento e move o ponteiro do cursor para o próximo elemento. |
| void | remove() | Ele remove os últimos elementos retornados pelo iterador. é menos usado. |

## Métodos da Classe CopyOnWriteArrayList

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Acrescenta o elemento especificado ao final desta lista. |
| void | add(int index, E element) | Insere o elemento especificado na posição especificada nesta lista. |
| boolean | addAll(Collection<? extends E> c) | Acrescenta todos os elementos na coleção especificada ao final desta lista, na ordem em que são retornados pelo iterador da coleção especificada. |
| boolean | addAll(int index, Collection<? extends E> c) | Insere todos os elementos da coleção especificada nesta lista, começando na posição especificada. |
| Int | addAllAbsent(Collection<? extends E> c) | Acrescenta todos os elementos na coleção especificada que ainda não estão contidos nesta lista, ao final desta lista, na ordem em que são retornados pelo iterador da coleção especificada. |
| Boolean | addIfAbsent(E e) | Acrescenta o elemento, se não estiver presente. |
| void | clear() | Remove todos os elementos desta lista. |
| Object | clone() | Retorna uma cópia rasa desta lista. |
| boolean | contains(Object o) | Retorna verdadeiro se esta lista contiver o elemento especificado. |
| boolean | containsAll(Collection<?> c) | Retorna verdadeiro se esta lista contiver todos os elementos da coleção especificada. |
| boolean | equals(Object o) | Compara o objeto especificado com esta lista para igualdade. |
| void | forEach(Consumer<? super E> action) | Executa a ação fornecida para cada elemento do Iterável até que todos os elementos tenham sido processados ​​ou a ação gere uma exceção. |
| E | get(int index) | Retorna o elemento na posição especificada nesta lista. |
| int | hashCode() | Retorna o valor do código hash para esta lista. |
| int | indexOf(E e, int index) | Retorna o índice da primeira ocorrência do elemento especificado nesta lista, pesquisando a partir do índice ou retorna -1 se o elemento não for encontrado. |
| int | indexOf(Object o) | Retorna o índice da primeira ocorrência do elemento especificado nesta lista ou -1 se esta lista não contiver o elemento. |
| boolean | isEmpty() | Retorna verdadeiro se esta lista não contém elementos. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos desta lista na sequência apropriada. |
| int | lastIndexOf(E e, int index) | Retorna o índice da última ocorrência do elemento especificado nesta lista, pesquisando para trás a partir do índice ou retorna -1 se o elemento não for encontrado. |
| int | lastIndexOf(Object o) | Retorna o índice da última ocorrência do elemento especificado nesta lista ou -1 se esta lista não contiver o elemento. |
| ListIterator<E> | listIterator() | Retorna um iterador de lista sobre os elementos dessa lista (na sequência adequada). |
| ListIterator<E> | listIterator(int index) | Retorna um iterador de lista sobre os elementos dessa lista (na sequência adequada), começando na posição especificada na lista. |
| E | remove(int index) | Remove o elemento na posição especificada nesta lista. |
| boolean | remove(Object o) | Remove a primeira ocorrência do elemento especificado desta lista, se estiver presente. |
| boolean | removeAll(Collection<?> c) | Remove desta lista todos os seus elementos contidos na coleção especificada. |
| boolean | removeIf(Predicate<? super E> filter) | Remove todos os elementos desta coleção que atendem ao predicado fornecido. |
| void | replaceAll(UnaryOperator<E> operator) | Substitui cada elemento desta lista pelo resultado da aplicação do operador a esse elemento. |
| boolean | retainAll(Collection<?> c) | Retém apenas os elementos desta lista que estão contidos na coleção especificada. |
| E | set(int index, E element) | Substitui o elemento na posição especificada nesta lista pelo elemento especificado. |
| int | size() | Retorna o número de elementos nesta lista. |
| void | sort(Comparator<? super E> c) | Classifica esta lista de acordo com a ordem induzida pelo Comparator especificado. |
| Spliterator<E> | spliterator() | Retorna um Spliterator sobre os elementos desta lista. |
| List<E> | subList(int fromIndex, int toIndex) | Retorna uma visão da parte desta lista entre fromIndex, inclusive, e toIndex, exclusivo. |
| Object[] | toArray() | Retorna um array contendo todos os elementos desta lista na sequência correta (do primeiro ao último elemento). |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos desta lista na sequência correta (do primeiro ao último elemento); o tipo de tempo de execução da matriz retornada é o da matriz especificada. |
| String | toString() | Retorna uma representação de string desta lista. |

## Métodos da Interface List

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | É usado para anexar o elemento especificado no final da Classe lista. |
| void | add(int index, E element) | É usado para inserir o elemento especificado na posição especificada em uma lista. |
| boolean | addAll(Collection<? extends E> c) | Ele é usado para anexar todos os elementos na coleção especificada ao final da Classe lista. |
| boolean | addAll(int index, Collection<? extends E> c) | Ele é usado para anexar todos os elementos na coleção especificada, começando na posição especificada da lista. |
| void | clear() | É usado para remover todos os elementos desta lista. |
| boolean | contains(Object o) | Retorna verdadeiro se a lista contiver o elemento especificado |
| boolean | containsAll(Collection<?> c) | Retorna verdadeiro se a lista contiver todos os elementos especificados |
| boolean | equals(Object o) | É usado para comparar o objeto especificado com os elementos da Classe lista. |
| E | get(int index) | É usado para buscar o elemento da posição específica da lista. |
| int | hashcode() | Ele é usado para retornar o valor do código hash para uma lista. |
| int | indexOf(Object o) | É utilizado para retornar o índice nesta lista da primeira ocorrência do elemento especificado, ou -1 se a Lista não contiver este elemento. |
| boolean | isEmpty() | Retorna verdadeiro se a lista estiver vazia, caso contrário, falso. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos nesta lista numa sequencia adequada. |
| int | lastIndexOf(Object o) | É usado para retornar o índice nesta lista da última ocorrência do elemento especificado, ou -1 se a lista não contiver este elemento. |
| ListIterator<E> | listIterator() | Retorna um iterador de lista sobre os elementos dessa lista (na sequência adequada). |
| ListIterator<E> | listIterator(int index) | Retorna um iterador de lista sobre os elementos dessa lista (na sequência adequada), começando na posição especificada na lista. |
| E | remove(int index) | É usado para remover o elemento presente na posição especificada na lista. |
| boolean | remove(Object o) | É usado para remover a primeira ocorrência do elemento especificado. |
| boolean | removeAll(Collection<?> c) | É usado para remover todos os elementos da lista. |
| default void | replaceAll(UnaryOperator<E> operator) | É usado para substituir todos os elementos da lista pelo elemento especificado. |
| boolean | retainAll(Collection<?> c) | Ele é usado para reter todos os elementos da lista que estão presentes na coleção especificada. |
| E | set(int index, E element) | É usado para substituir o elemento especificado na lista, presente na posição especificada. |
| int | size() | É usado para retornar o número de elementos presentes na lista. |
| default void | sort(Comparator<? super E> c) | É usado para classificar os elementos da lista com base no comparador especificado. |
| default Spliterator<E> | spliterator() | É usado para criar spliterator sobre os elementos em uma lista. |
| List<E> | subList(int fromIndex, int toIndex) | É usado para buscar todos os elementos dentro da Classe determinado intervalo. |
| Object[] | toArray() | Ele é usado para retornar um array contendo todos os elementos desta lista na ordem correta. |
| <T> T[] | toArray(T[] a) | Ele é usado para retornar um array contendo todos os elementos desta lista na ordem correta. |

## Métodos da Classe LinkedList

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | É usado para anexar o elemento especificado ao final da Classe lista. |
| void | add(int index, E element) | É usado para inserir o elemento especificado no índice de posição especificado em uma lista. |
| boolean | addAll(Collection<? extends E> c) | Ele é usado para anexar todos os elementos na coleção especificada ao final desta lista, na ordem em que são retornados pelo iterador da coleção especificada. |
| boolean | addAll(Collection<? extends E> c) | Ele é usado para anexar todos os elementos na coleção especificada ao final desta lista, na ordem em que são retornados pelo iterador da coleção especificada. |
| boolean | addAll(int index, Collection<? extends E> c) | Ele é usado para anexar todos os elementos na coleção especificada, começando na posição especificada da lista. |
| void | addFirst(E e) | É usado para inserir o elemento dado no início da Classe lista. |
| void | addLast(E e) | É usado para anexar o elemento fornecido ao final da Classe lista. |
| void | clear() | É usado para remover todos os elementos da Classe lista. |
| Object | clone() | É usado para retornar uma cópia superficial da Classe ArrayList. |
| boolean | contains(Object o) | É usado para retornar verdadeiro se uma lista contiver um elemento especificado. |
| Iterator<E> | descendingIterator() | É usado para retornar um iterador sobre os elementos em um deque em ordem sequencial reversa. |
| E | element() | É usado para recuperar o primeiro elemento da Classe lista. |
| E | get(int index) | É usado para retornar o elemento na posição especificada em uma lista. |
| E | getFirst() | É usado para retornar o primeiro elemento da Classe lista. |
| E | getLast() | É usado para retornar o último elemento da Classe lista. |
| int | indexOf(Object o) | É usado para retornar o índice em uma lista da primeira ocorrência do elemento especificado, ou -1 se a lista não contiver nenhum elemento. |
| int | lastIndexOf(Object o) | É usado para retornar o índice em uma lista da última ocorrência do elemento especificado, ou -1 se a lista não contiver nenhum elemento. |
| ListIterator<E> | listIterator(int index) | Ele é usado para retornar um iterador de lista dos elementos na sequência adequada, começando na posição especificada na lista. |
| boolean | offer(E e) | Ele adiciona o elemento especificado como o último elemento da Classe lista. |
| boolean | offerFirst(E e) | Ele insere o elemento especificado na frente da Classe lista. |
| boolean | offerLast(E e) | Ele insere o elemento especificado no final da Classe lista. |
| E | peek() | Ele recupera o primeiro elemento da Classe lista |
| E | peekFirst() | Ele recupera o primeiro elemento da Classe lista ou retorna null se a lista estiver vazia. |
| E | peekLast() | Ele recupera o último elemento da Classe lista ou retorna null se uma lista estiver vazia. |
| E | poll() | Ele recupera e remove o primeiro elemento da Classe lista. |
| E | pollFirst() | Ele recupera e remove o primeiro elemento da Classe lista ou retorna null se uma lista estiver vazia. |
| E | pollLast() | Ele recupera e remove o último elemento da Classe lista ou retorna null se uma lista estiver vazia. |
| E | pop() | Ele extrai um elemento da pilha representada por uma lista. |
| void | push(E e) | Ele empurra um elemento para a pilha representada por uma lista. |
| E | remove() | Ele é usado para recuperar e remover o primeiro elemento da Classe lista. |
| E | remove(int index) | É usado para remover o elemento na posição especificada em uma lista. |
| boolean | remove(Object o) | É usado para remover a primeira ocorrência do elemento especificado em uma lista. |
| E | removeFirst() | Ele remove e retorna o primeiro elemento da Classe lista. |
| boolean | removeFirstOccurrence(Object o) | É usado para remover a primeira ocorrência do elemento especificado em uma lista (ao percorrer a lista do início ao fim). |
| E | removeLast() | Ele remove e retorna o último elemento da Classe lista. |
| boolean | removeLastOccurrence(Object o) | Ele remove a última ocorrência do elemento especificado em uma lista (ao percorrer a lista do início ao fim). |
| E | set(int index, E element) | Ele substitui o elemento na posição especificada em uma lista pelo elemento especificado. |
| int | size() | É usado para retornar o número de elementos em uma lista. |
| Object[] | toArray() | É usado para retornar um array contendo todos os elementos da Classe lista na sequência correta (do primeiro ao último elemento). |
| <T> T[] | toArray(T[] a) | Ele retorna um array contendo todos os elementos na sequência correta (do primeiro ao último elemento); o tipo de tempo de execução da matriz retornada é o da matriz especificada. |

## Métodos da Classe Vetor

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Acrescenta o elemento especificado ao final deste Vector. |
| void | add(int index, E element) | Insere o elemento especificado na posição especificada neste Vetor. |
| boolean | addAll(Collection<?  extends E> c) | Acrescenta todos os elementos na Coleção especificada ao final deste Vetor, na ordem em que são retornados pelo Iterador da Coleção especificada. |
| boolean | addAll(int index,  Collection<? extends E> c) | Insira todos os elementos na Coleção especificada neste Vetor na posição especificada. |
| void | addElement(E obj) | Adiciona o componente especificado ao final deste vetor, aumentando seu tamanho em um. |
| int | capacity() | Retorna a capacidade atual deste vetor. |
| void | clear() | Remove todos os elementos deste Vector. |
| Object | clone() | Retorna um clone deste vetor. |
| boolean | contains(Object o) | Retorna verdadeiro se este vetor contiver o elemento especificado. |
| boolean | containsAll(Collection<?> c) | Retorna verdadeiro se este Vetor contiver todos os elementos da Coleção especificada. |
| void | copyInto(Object[] anArray) | Copia os componentes deste vetor na matriz especificada. |
| E | elementAt(int index) | Retorna o componente no índice especificado. |
| Enumeration<E> | elements() | Retorna uma enumeração dos componentes deste vetor. |
| void | ensureCapacity(int minCapacity) | Aumenta a capacidade desse vetor, se necessário, para garantir que ele possa conter pelo menos o número de componentes especificado pelo argumento de capacidade mínima. |
| boolean | equals(Object o) | Compara o Object especificado com este Vector para igualdade. |
| E | firstElement() | Retorna o primeiro componente (o item no índice 0) deste vetor. |
| void | forEach(Consumer<?  super E> action) | Executa a ação fornecida para cada elemento do Iterável até que todos os elementos tenham sido processados ​​ou a ação lance uma exceção. |
| E | get(int index) | Retorna o elemento na posição especificada neste Vector. |
| int | hashCode() | Retorna o valor do código hash para este Vector. |
| int | indexOf(Object o) | Retorna o índice da primeira ocorrência do elemento especificado neste vetor, |
| int | indexOf(Object o, int index) | ou -1 se este vetor não contém o elemento. |
| void | insertElementAt(E obj, int index) | Retorna o índice da primeira ocorrência do elemento especificado neste vetor, pesquisando a partir do índice, ou retorna -1 se o elemento não for encontrado. |
| boolean | isEmpty() | Insere o objeto especificado como um componente neste vetor no índice especificado. |
| Iterator<E> | iterator() | Testa se este vetor não tem componentes. |
| E | lastElement() | Retorna um iterador sobre os elementos desta lista em uma sequência adequada. |
| int | lastIndexOf(Object o) | Retorna o último componente do vetor. |
| int | lastIndexOf(Object o, int index) | Retorna o índice da última ocorrência do elemento especificado neste vetor, |
| ListIterator<E> | listIterator() | ou -1 se este vetor não contém o elemento. |
| ListIterator<E> | listIterator(int index) | Retorna o índice da última ocorrência do elemento especificado neste vetor, pesquisando para trás a partir do índice, ou retorna -1 se o elemento não for encontrado. |
| E | remove(int index) | Remove o elemento na posição especificada neste Vector. |
| boolean | remove(Object o) | Remove a primeira ocorrência do elemento especificado neste Vector. Se o Vetor não contiver o elemento, ele permanecerá inalterado. |
| boolean | removeAll(Collection<?> c) | Remove deste Vetor todos os seus elementos contidos na Coleção especificada. |
| void | removeAllElements() | Remove todos os componentes deste vetor e define seu tamanho como zero. |
| boolean | removeElement(Object obj) | Remove a primeira ocorrência (menor índice) do argumento deste vetor. |
| void | removeElementAt(int index) | Exclui o componente no índice especificado. |
| boolean | removeIf(Predicate<? super E> filter) | Remove todos os elementos desta coleção que atendem ao predicado fornecido. |
| protected void | removeRange(int fromIndex,  int toIndex) | Remove desta lista todos os elementos cujo índice está entre fromIndex, inclusive, e toIndex, exclusivo. |
| void | replaceAll(UnaryOperator<E> operator) | Substitui cada elemento desta lista pelo resultado da aplicação do operador a esse elemento. |
| boolean | retainAll(Collection<?> c) | Retém apenas os elementos neste Vetor contidos na Coleção especificada. |
| E | set(int index, E element) | Substitui o elemento na posição especificada neste Vetor pelo elemento especificado. |
| void | setElementAt(E obj, int index) | Define o componente no índice especificado deste vetor para ser o objeto especificado. |
| void | setSize(int newSize) | Define o tamanho deste vetor. |
| int | size() | Retorna o número de componentes neste vetor. |
| void | sort(Comparator<? super E> c) | Classifica esta lista de acordo com a ordem induzida pelo Comparator especificado. |
| Spliterator<E> | spliterator() | Cria um Spliterator de ligação tardia e falha rápida sobre os elementos nesta lista. |
| List<E> | subList(int fromIndex, int toIndex) | Retorna uma visão da parte desta Lista entre fromIndex, inclusive, e toIndex, exclusivo. |
| Object[] | toArray() | Retorna um array contendo todos os elementos deste Vector na ordem correta. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos deste Vector na ordem correta; o tipo de tempo de execução da matriz retornada é o da matriz especificada. |
| String | toString() | Retorna uma representação em string deste Vetor, contendo a representação em String de cada elemento. |
| void | trimToSize() | Ajusta a capacidade deste vetor para o tamanho atual do vetor. |

## Métodos da Classe Stack

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | empty() | O método empurra (inserir) um elemento para o topo da pilha. |
| E | push(E item) | O método remove um elemento do topo da pilha e retorna o mesmo elemento que o valor dessa função. |
| E | pop() | O método examina o elemento superior da pilha sem removê-lo. |
| E | peek() | O método pesquisa o objeto especificado e retorna a posição do objeto. |
| int | search(Object o) | O método empurra (inserir) um elemento para o topo da pilha. |

## Métodos da Classe ArrayList

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | É usado para anexar o elemento especificado no final da Classe lista. |
| void | add(int index, E element) | É usado para inserir o elemento especificado na posição especificada em uma lista. |
| boolean | addAll(Collection<? extends E> c) | Ele é usado para anexar todos os elementos na coleção especificada ao final desta lista, na ordem em que são retornados pelo iterador da coleção especificada. |
| boolean | addAll(int index, Collection<? extends E> c) | Ele é usado para anexar todos os elementos na coleção especificada, começando na posição especificada da lista. |
| void | clear() | É usado para remover todos os elementos desta lista. |
| Object | clone() | É usado para retornar uma cópia superficial da Classe ArrayList. |
| boolean | contains(Object o) | Retorna verdadeiro se a lista contiver o elemento especificado. |
| void | ensureCapacity(int requiredCapacity) | Ele é usado para aumentar a capacidade da Classe instância ArrayList. |
| E | get(int index) | É usado para buscar o elemento da posição específica da lista. |
| int | indexOf(Object o) | É utilizado para retornar o índice nesta lista da primeira ocorrência do elemento especificado, ou -1 se a Lista não contiver este elemento. |
| boolean | isEmpty() | Retorna verdadeiro se a lista estiver vazia, caso contrário, falso. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos desta lista. |
| int | lastIndexOf(Object o) | É usado para retornar o índice nesta lista da última ocorrência do elemento especificado, ou -1 se a lista não contiver este elemento. |
| ListIterator<E> | listIterator() | Retorna o iterador de lista sobre os elementos nesta lista começando na posição especificada nesta lista. |
| E | remove(int index) | É usado para remover o elemento presente na posição especificada na lista. |
| boolean | remove(Object o) | É usado para remover a primeira ocorrência do elemento especificado. |
| boolean | removeAll(Collection<?> c) | É usado para remover todos os elementos da lista. |
| boolean | removeIf(Predicate<? super E> filter) | É usado para remover todos os elementos da lista que satisfazem o predicado fornecido. |
| protected void | removeRange(int fromIndex, int toIndex) | É usado para remover todos os elementos dentro do intervalo especificado. |
| void | replaceAll(UnaryOperator<E> operator) | É usado para substituir todos os elementos da lista pelo elemento especificado. |
| void | retainAll(Collection<?> c) | Ele é usado para reter todos os elementos da lista que estão presentes na coleção especificada. |
| E | set(int index, E element) | É usado para substituir o elemento especificado na lista, presente na posição especificada. |
| void | sort(Comparator<? super E> c) | Ele é usado para classificar os elementos da lista com base no comparador especificado. |
| Spliterator<E> | spliterator() | Ele é usado para criar um divisor sobre os elementos da Classe lista. |
| Object[] | toArray() | Ele é usado para retornar um array contendo todos os elementos desta lista na ordem correta. |
| <T> T[] | toArray(T[] a) | Ele é usado para retornar um array contendo todos os elementos desta lista na ordem correta. |

## Métodos da Interface Queue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Metodos | Descrição |
| boolean | add(E e) | Adiciona o elemento e à fila no final (final) da fila sem violar as restrições de capacidade. Retorna true se for bem-sucedido ou IllegalStateException se a capacidade for esgotada. |
| E | element() | Retorna, mas não remove o primeiro elemento desta fila. |
| boolean | offer(E e) | Insere o elemento especificado nesta fila se for possível feito imediatamente sem violar as restrições de capacidade. |
| E | peek() | Retorna, mas não remove o primeiro elemento desta fila ou retorna nulo se esta fila estiver vazia. |
| E | poll() | Retorna e remove o primeiro elemento desta fila ou retorna nulo se esta fila estiver vazia. |
| E | remove() | Retorna e remove o primeiro elemento desta fila. |

## Métodos da Classe ConcurrentLinkedQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado no final desta fila |
| boolean | addAll(Collection<? Extends E> c) | Acrescenta todos os elementos na coleção especificada ao final desta fila, na ordem em que são retornados pelo iterador da coleção especificada. |
| boolean | contains(Object o) | Retorna verdadeiro se esta fila contiver o elemento especificado. |
| boolean | isEmpty() | Retorna verdadeiro se esta fila não contém elementos. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos desta fila na sequência adequada. |
| boolean | offer(E e) | Insere o elemento especificado no final desta fila. |
| E | peek() | Recupera, mas não remove, o cabeçalho desta fila ou retorna nulo se esta fila estiver vazia. |
| E | poll() | Recupera e remove o início desta fila ou retorna nulo se esta fila estiver vazia. |
| boolean | remove(Object o) | Remove uma única instância do elemento especificado desta fila, se estiver presente. |
| int | size() | Retorna o número de elementos nesta fila. |
| Spliterator<E> | spliterator() | Retorna um Spliterator sobre os elementos desta fila. |
| Object[] | toArray() | Retorna um array contendo todos os elementos desta fila, na sequência correta. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos desta fila, na sequência correta; o tipo de tempo de execução da matriz retornada é o da matriz especificada. |

## Métodos da Classe ProrityQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Adiciona o elemento especificado dentro desta priority queue. |
| void | clear() | Remove todos os elementos desta priority queue. |
| Comparator<? super E> | comparator() | Retorna um comparador usado para ordenar os elementos nesta fila, ou retorna null se os elementos desta lista estiverem ordenados de forma natural de acordo com os elementos. |
| boolean | contains(Object o) | Retorna true se esta fila contem o elemento especificado |
| Interator<E> | iterator() | Retorna um iterador sobre os elementos desta lista. |
| boolean | offer(E e) | Insere o elemento especificado nesta priority queue se for possível feito imediatamente sem violar as restrições de capacidade. |
| E | peek() | Retorna, mas não remove o primeiro elemento desta fila ou retorna nulo se esta fila estiver vazia. |
| E | poll() | Retorna e remove o primeiro elemento desta fila ou retorna nulo se esta fila estiver vazia. |
| boolean | remove() | Retorna e remove o primeiro elemento desta priority queue. |
| int | size() | Retorna o numero de elementos desta coleção. |
| Object[] | toArray() | Retorna uma array contento todos os elementos desta fila. |
| <T> T[] | toArray(T[] a) | Retorna uma array contento todos os elementos desta fila; o tipo de tempo de execução da array retornada é o da array especificada. |

## Métodos da Interface Blocking Queue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado nesta fila se for possível fazê-lo imediatamente sem violar as restrições de capacidade, retornando true em caso de sucesso e lançando um IllegalStateException se não houver espaço disponível no momento. |
| boolean | contains(Object o) | Retorna verdadeiro se esta fila contiver o elemento especificado. |
| int | drainTo(Collenction<? super E> c) | Remove todos os elementos disponíveis desta fila e os adiciona à coleção especificada. |
| int | drainTo(Collenction<? super E> c, int maxElements) | Remove no máximo o número fornecido de elementos disponíveis desta fila e os adiciona à coleção especificada. |
| boolean | offer(E e) | Insere o elemento especificado nesta fila se for possível fazê-lo imediatamente sem violar as restrições de capacidade, retornando verdadeiro em caso de sucesso e falso se não houver espaço disponível no momento. |
| boolean | offer(E e, long timeout, TimeUnit unit) | Insere o elemento especificado nessa fila, aguardando até o tempo de espera especificado, se necessário, para que o espaço fique disponível. |
| E | poll(long timeout, TimeUnit unit) | Recupera e remove o início desta fila, aguardando até o tempo de espera especificado, se necessário, para que um elemento fique disponível. |
| void | put(E e) | Insere o elemento especificado nesta fila, esperando, se necessário, que o espaço fique disponível. |
| int | remainingCapacity() | Retorna o número de elementos adicionais que esta fila pode idealmente (na ausência de restrições de memória ou recursos) aceitar sem bloqueio, ou Integer.MAX\_VALUE se não houver limite intrínseco. |
| boolean | remove(Object o) | Remove uma única instância do elemento especificado desta fila, se estiver presente. |
| E | take() | Recupera e remove o cabeçalho desta fila, esperando se necessário até que um elemento fique disponível. |

## Métodos da Classe ArrayBlockingQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado no final da fila. |
| void | clear() | Remove todos os elementos da fila. |
| boolean | contains(Object o) | Retorna true se a fila contém o elemento especificado. |
| int | drainTo(Collection<? super E> c) | Remove todos os elementos da fila e os adiciona à coleção especificada. |
| int | drainTo(Collection<? super E> c, int maxElements) | Remove no máximo o número especificado de elementos da fila e os adiciona à coleção especificada. |
| E | element() | Retorna o primeiro elemento da fila sem removê-lo. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos da fila. |
| boolean | offer(E e) | Insere o elemento especificado no final da fila. |
| boolean | offer(E e, long timeout, TimeUnit unit) | Insere o elemento especificado no final da fila, esperando o tempo máximo especificado se necessário. |
| E | poll() | Remove e retorna o primeiro elemento da fila, ou retorna null se a fila estiver vazia. |
| E | poll(long timeout, TimeUnit unit) | Remove e retorna o primeiro elemento da fila, esperando o tempo máximo especificado se necessário. |
| void | put(E e) | Insere o elemento especificado no final da fila, esperando se necessário até que haja espaço disponível. |
| int | remainingCapacity() | Retorna a quantidade de espaço disponível na fila. |
| boolean | remove(Object o) | Remove a primeira ocorrência do elemento especificado da fila, se presente. |
| int | size() | Retorna o número de elementos na fila. |
| E | take() | Remove e retorna o primeiro elemento da fila, esperando se necessário até que haja um elemento disponível. |
| Object[] | toArray() | Retorna um array contendo todos os elementos da fila, na ordem em que foram inseridos. |
| T[] | toArray(T[] a) | Retorna um array contendo todos os elementos da fila, na ordem em que foram inseridos, e com o tipo de elemento especificado. |

## Métodos da Classe LinkedBlockingQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado no final da fila. |
| void | clear() | Remove todos os elementos da fila. |
| boolean | contains(Object o) | Retorna true se a fila contém o elemento especificado. |
| int | drainTo(Collection<? super E> c) | Remove todos os elementos da fila e os adiciona à coleção especificada. |
| int | drainTo(Collection<? super E> c, int maxElements) | Remove no máximo o número especificado de elementos da fila e os adiciona à coleção especificada. |
| E | element() | Retorna o primeiro elemento da fila sem removê-lo. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos da fila. |
| boolean | offer(E e) | Insere o elemento especificado no final da fila. |
| boolean | offer(E e, long timeout, TimeUnit unit) | Insere o elemento especificado no final da fila, esperando o tempo máximo especificado se necessário. |
| E | poll() | Remove e retorna o primeiro elemento da fila, ou retorna null se a fila estiver vazia. |
| E | poll(long timeout, TimeUnit unit) | Remove e retorna o primeiro elemento da fila, esperando o tempo máximo especificado se necessário. |
| void | put(E e) | Insere o elemento especificado no final da fila, esperando se necessário até que haja espaço disponível. |
| int | remainingCapacity() | Retorna a quantidade de espaço disponível na fila (sempre Integer.MAX\_VALUE). |
| boolean | remove(Object o) | Remove a primeira ocorrência do elemento especificado da fila, se presente. |
| int | size() | Retorna o número de elementos na fila. |
| E | take() | Remove e retorna o primeiro elemento da fila, esperando se necessário até que haja um elemento disponível. |
| Object[] | toArray() | Retorna um array contendo todos os elementos da fila, na ordem em que foram inseridos. |
| T[] | toArray(T[] a) | Retorna um array contendo todos os elementos da fila, na ordem em que foram inseridos, e com o tipo de elemento especificado. |

## Métodos da Classe SynchronousQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| int | drainTo(Collection<? super E> c) | Remove todos os elementos da fila e os adiciona à coleção especificada (o número de elementos removidos será sempre 0 ou 1). |
| int | drainTo(Collection<? super E> c, int maxElements) | Remove no máximo o número especificado de elementos da fila e os adiciona à coleção especificada (o número de elementos removidos será sempre 0 ou 1). |
| boolean | offer(E e) | Insere o elemento especificado na fila, se houver um consumidor esperando para recebê-lo; caso contrário, retorna false. |
| E | poll() | Remove e retorna o elemento da fila, se houver um produtor esperando para inserir um elemento; caso contrário, retorna null. |
| void | put(E e) | Insere o elemento especificado na fila, esperando se necessário até que um consumidor esteja pronto para recebê-lo. |
| E | take() | Remove e retorna o elemento da fila, esperando se necessário até que um produtor esteja pronto para inserir um elemento. |
| boolean | tryAcquireShared(int arg) | Semelhante ao método tryAcquire na classe AbstractQueuedSynchronizer, mas sempre retorna false. |
| boolean | tryReleaseShared(int arg) | Semelhante ao método tryRelease na classe AbstractQueuedSynchronizer, mas sempre retorna false. |

## Métodos da Classe PriorityBlockingQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado na fila. |
| void | clear() | Remove todos os elementos da fila. |
| Comparator<? super E> | comparator() | Retorna o comparador usado para ordenar os elementos da fila, ou null se a ordem natural for usada. |
| boolean | contains(Object o) | Retorna true se a fila contiver o elemento especificado. |
| void | drainTo(Collection<? super E> c) | Remove todos os elementos da fila e os adiciona à coleção especificada. |
| void | drainTo(Collection<? super E> c, int maxElements) | Remove no máximo o número especificado de elementos da fila e os adiciona à coleção especificada. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos na fila. |
| boolean | offer(E e) | Insere o elemento especificado na fila. |
| E | peek() | Recupera, mas não remove, o elemento na frente da fila. |
| E | poll() | Recupera e remove o elemento na frente da fila. |
| void | put(E e) | Insere o elemento especificado na fila, esperando se necessário até que haja espaço disponível. |
| boolean | remove(Object o) | Remove o elemento especificado da fila, se estiver presente. |
| int | size() | Retorna o número de elementos na fila. |
| Object[] | toArray() | Retorna um array contendo todos os elementos na fila. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos na fila; o tipo de tempo de execução do array retornado é o do array especificado. |

## Métodos da Classe DelayQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado na fila. |
| void | clear() | Remove todos os elementos da fila. |
| Comparator<? super Delayed> | comparator() | Retorna o comparador usado para ordenar os elementos da fila, ou null se a ordem natural for usada. |
| boolean | contains(Object o) | Retorna true se a fila contiver o elemento especificado. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos na fila. |
| boolean | offer(E e) | Insere o elemento especificado na fila. |
| E | peek() | Recupera, mas não remove, o elemento na frente da fila. |
| E | poll() | Recupera e remove o elemento na frente da fila. |
| void | put(E e) | Insere o elemento especificado na fila, esperando se necessário até que haja espaço disponível. |
| boolean | remove(Object o) | Remove o elemento especificado da fila, se estiver presente. |
| int | size() | Retorna o número de elementos na fila. |
| Object[] | toArray() | Retorna um array contendo todos os elementos na fila. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos na fila; o tipo de tempo de execução do array retornado é o do array especificado. |
| void | drainTo(Collection<? super E> c) | Remove todos os elementos da fila e os adiciona à coleção especificada. |
| void | drainTo(Collection<? super E> c, int maxElements) | Remove no máximo o número especificado de elementos da fila e os adiciona à coleção especificada. |
| long | remainingDelay(TimeUnit unit) | Retorna a quantidade de tempo restante até que o próximo elemento possa ser removido da fila. |
| E | take() | Recupera e remove o elemento na frente da fila, esperando se necessário até que a sua data/hora de expiração tenha sido alcançada. |

## Métodos da Interface TransferQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado na fila. |
| boolean | tryAdd(E e, long timeout, TimeUnit unit) | Insere o elemento especificado na fila, aguardando no máximo o tempo especificado. |
| boolean | tryTransfer(E e) | Insere o elemento especificado na fila, se possível, sem bloquear. |
| boolean | tryTransfer(E e, long timeout, TimeUnit unit) | Insere o elemento especificado na fila, aguardando no máximo o tempo especificado. |
| void | put(E e) | Insere o elemento especificado na fila, aguardando se necessário até que haja espaço disponível. |
| boolean | offer(E e) | Insere o elemento especificado na fila. |
| boolean | offer(E e, long timeout, TimeUnit unit) | Insere o elemento especificado na fila, aguardando no máximo o tempo especificado. |
| E | take() | Recupera e remove o elemento na frente da fila, esperando se necessário até que esteja disponível. |
| E | poll() | Recupera e remove o elemento na frente da fila, se estiver disponível. |
| E | poll(long timeout, TimeUnit unit) | Recupera e remove o elemento na frente da fila, aguardando no máximo o tempo especificado. |
| int | drainTo(Collection<? super E> c) | Remove todos os elementos da fila e os adiciona à coleção especificada. |
| int | drainTo(Collection<? super E> c, int maxElements) | Remove no máximo o número especificado de elementos da fila e os adiciona à coleção especificada. |
| boolean | hasWaitingConsumer() | Retorna true se há consumidores aguardando para receber um elemento. |
| int | getWaitingConsumerCount() | Retorna o número de consumidores aguardando para receber um elemento. |
| boolean | tryTransfer(E e) | Tenta transferir o elemento especificado para um consumidor, se houver um esperando. |
| E | transfer(E e) | Transfere o elemento especificado para um consumidor, aguardando se necessário até que haja um consumidor disponível. |
| int | getWaitingConsumerCount() | Retorna o número de consumidores aguardando para receber um elemento. |
| boolean | tryTransfer(E e, long timeout, TimeUnit unit) | Tenta transferir o elemento especificado para um consumidor, aguardando no máximo o tempo especificado. |
| boolean | tryTransfer(E e, long timeout, TimeUnit unit) | Tenta transferir o elemento especificado para um consumidor, aguardando no máximo o tempo especificado. |

## Métodos da Classe LinkedTransferQueue

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado na fila. |
| boolean | tryAdd(E e, long timeout, TimeUnit unit) | Insere o elemento especificado na fila, aguardando no máximo o tempo especificado. |
| boolean | tryTransfer(E e) | Insere o elemento especificado na fila, se possível, sem bloquear. |
| boolean | tryTransfer(E e, long timeout, TimeUnit unit) | Insere o elemento especificado na fila, aguardando no máximo o tempo especificado. |
| void | put(E e) | Insere o elemento especificado na fila, aguardando se necessário até que haja espaço disponível. |
| boolean | offer(E e) | Insere o elemento especificado na fila. |
| boolean | offer(E e, long timeout, TimeUnit unit) | Insere o elemento especificado na fila, aguardando no máximo o tempo especificado. |
| E | take() | Recupera e remove o elemento na frente da fila, esperando se necessário até que esteja disponível. |
| E | poll() | Recupera e remove o elemento na frente da fila, se estiver disponível. |
| E | poll(long timeout, TimeUnit unit) | Recupera e remove o elemento na frente da fila, aguardando no máximo o tempo especificado. |
| int | drainTo(Collection<? super E> c) | Remove todos os elementos da fila e os adiciona à coleção especificada. |
| int | drainTo(Collection<? super E> c, int maxElements) | Remove no máximo o número especificado de elementos da fila e os adiciona à coleção especificada. |
| boolean | hasWaitingConsumer() | Retorna true se há consumidores aguardando para receber um elemento. |
| int | getWaitingConsumerCount() | Retorna o número de consumidores aguardando para receber um elemento. |
| boolean | tryTransfer(E e) | Tenta transferir o elemento especificado para um consumidor, se houver um esperando. |
| E | transfer(E e) | Transfere o elemento especificado para um consumidor, aguardando se necessário até que haja um consumidor disponível. |
| int | getWaitingConsumerCount() | Retorna o número de consumidores aguardando para receber um elemento. |
| boolean | tryTransfer(E e, long timeout, TimeUnit unit) | Tenta transferir o elemento especificado para um consumidor, aguardando no máximo o tempo especificado. |
| boolean | tryTransfer(E e, long timeout, TimeUnit unit) | Tenta transferir o elemento especificado para um consumidor, aguardando no máximo o tempo especificado. |

## Métodos da Interface Deque

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Este método é usado para adicionar um elemento no final da fila. Se o Deque tiver capacidade restrita e não houver espaço para inserção, ele retornará um IllegalStateException. A função retorna true se a inserção for bem-sucedida. |
| void | addFirst(E e) | Este método é usado para adicionar um elemento no início da fila. Se o Deque tiver capacidade restrita e não houver espaço para inserção, ele retornará um IllegalStateException. A função retorna true se a inserção for bem-sucedida. |
| boolean | addLast(E e) | Este método é usado para adicionar um elemento no final da fila. Se o Deque tiver capacidade restrita e não houver espaço para inserção, ele retornará um IllegalStateException. A função retorna true se a inserção for bem-sucedida. |
| boolean | contains() | Este método é usado para verificar se a fila contém o objeto fornecido ou não. |
| void | descendingIterator() | Este método retorna um iterador para o deque. Os elementos serão retornados em ordem crescente, do ultimo ao primeiro. |
| int | element() | Este método é usado para recuperar, mas não remover, a cabeça da fila representada por este deque. |
| void | getFirst() | Este método é usado para recuperar, mas não remover, o primeiro elemento deste deque. |
| Object | getLast() | Este método é usado para recuperar, mas não remover, o último elemento deste deque. |
| boolean | iterator() | Este método retorna um iterador para o deque. Os elementos serão retornados em ordem descrescente, do primeiro ao ultimo. |
| boolean | offer(element) | Insere o elemento especificado no final deste deque, a menos que viole as restrições de capacidade. É preferível ao método add() pois não lança uma exceção. Retorna false se a inserção não for bem sucedida. |
| void | offerFirst(element) | Insere o elemento especificado no final deste deque, a menos que viole as restrições de capacidade. É preferível ao método addFirst() pois não lança uma exceção. Retorna false se a inserção não for bem sucedida. |
| E | offerLast(element) | Insere o elemento especificado no final deste deque, a menos que viole as restrições de capacidade. É preferível ao método add() pois não lança uma exceção. Retorna false se a inserção não for bem sucedida. |
| Enumeration<E> | peek() | Este método é usado para recuperar o elemento no cabeçalho do deque, mas não remove o elemento deste deque. Este método retorna null se o deque estiver vazio. |
| void | peekFirst() | Este método é usado para recuperar o elemento no cabeçalho do deque, mas não remove o elemento deste deque. Este método retorna null se o deque estiver vazio. |
| boolean | peekLast() | Este método é usado para recuperar o elemento no final do deque, mas não remove o elemento deste deque. Este método retorna null se o deque estiver vazio. |
| E | poll() | Este método é usado para recuperar e remover o elemento no início do deque. Este método retorna null se o deque estiver vazio. |
| void | pollFirst() | Este método é usado para recuperar e remover o elemento no início deste deque. Este método retorna null se o deque estiver vazio. |
| E | pollLast() | Este método é usado para recuperar e remover o elemento no final deste deque. Este método retorna null se o deque estiver vazio. |
| int | pop() | Este método é usado para remover um elemento da cabeça e retorná-lo. |
| int | push(element) | Este método é usado para adicionar um elemento no início deste deque. |
| int | removeFirst() | Este método é usado para remover um elemento do início deste deque. |
| void | removeLast() | Este método é usado para remover um elemento do final deste deque. |
| boolean | size() | Este método é usado para localizar e retornar o tamanho deste deque. |

## Métodos da Classe ArrayDeque

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado no final da deque. |
| void | addFirst(E e) | Insere o elemento especificado no início da deque. |
| void | addLast(E e) | Insere o elemento especificado no final da deque. |
| void | clear() | Remove todos os elementos da deque. |
| Iterator<E> | descendingIterator() | Retorna um iterador |
| E | element() | Retorna o elemento no início da deque, sem removê-lo. |
| E | getFirst() | Retorna o elemento no início da deque, sem removê-lo. |
| E | getLast() | Retorna o elemento no final da deque, sem removê-lo. |
| boolean | isEmpty() | Retorna true se a deque estiver vazia, false caso contrário. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos na deque (da frente para trás). |
| boolean | offer(E e) | Insere o elemento especificado no final da deque. |
| boolean | offerFirst(E e) | Insere o elemento especificado no início da deque. |
| boolean | offerLast(E e) | Insere o elemento especificado no final da deque. |
| E | peek() | Retorna o elemento no início da deque, ou retorna null se a deque estiver vazia. |
| E | peekFirst() | Retorna o elemento no início da deque, ou retorna null se a deque estiver vazia. |
| E | peekLast() | Retorna o elemento no final da deque, ou retorna null se a deque estiver vazia. |
| E | poll() | Remove e retorna o elemento no início da deque, ou retorna null se a deque estiver vazia. |
| E | pollFirst() | Remove e retorna o elemento no início da deque, ou retorna null se a deque estiver vazia. |
| E | pollLast() | Remove e retorna o elemento no final da deque, ou retorna null se a deque estiver vazia. |
| E | pop() | Remove e retorna o elemento no início da deque. |
| void | push(E e) | Insere o elemento especificado no início da deque. |
| E | remove() | Remove e retorna o elemento no início da deque. |
| E | removeFirst() | Remove e retorna o elemento no início da deque. |
| boolean | removeFirstOccurrence(Object o) | Remove a primeira ocorrência do elemento especificado na deque (da frente para trás). |
| E | removeLast() | Remove e retorna o elemento no final da deque. |
| boolean | removeLastOccurrence(Object o) | Remove a última ocorrência do elemento especificado na deque (de trás para frente). |
| int | size() | Retorna o número de elementos na deque. |

## Métodos da Classe ConcurrentLinkedDeque

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | addFirst | Insere o elemento especificado no início da deque. |
| void | addLast | Insere o elemento especificado no final da deque. |
| void | clear | Remove todos os elementos da deque. |
| boolean | contains | Retorna true se a deque contém o elemento especificado. |
| Iterator<E> | descendingIterator | Retorna um iterador sobre os elementos na ordem reversa. |
| E | element | Retorna o primeiro elemento na deque sem removê-lo. |
| E | getFirst | Retorna o primeiro elemento na deque sem removê-lo. |
| E | getLast | Retorna o último elemento na deque sem removê-lo. |
| boolean | isEmpty | Retorna true se a deque não contém nenhum elemento. |
| Iterator<E> | iterator | Retorna um iterador sobre os elementos na ordem em que foram adicionados. |
| boolean | offer | Insere o elemento especificado no final da deque. |
| boolean | offerFirst | Insere o elemento especificado no início da deque. |
| boolean | offerLast | Insere o elemento especificado no final da deque. |
| E | peek | Retorna o primeiro elemento na deque, ou null se estiver vazia. |
| E | peekFirst | Retorna o primeiro elemento na deque, ou null se estiver vazia. |
| E | peekLast | Retorna o último elemento na deque, ou null se estiver vazia. |
| E | poll | Retorna e remove o primeiro elemento da deque, ou null se estiver vazia. |
| E | pollFirst | Retorna e remove o primeiro elemento da deque, ou null se estiver vazia. |
| E | pollLast | Retorna e remove o último elemento da deque, ou null se estiver vazia. |
| E | pop | Remove e retorna o primeiro elemento da deque. |
| void | push | Insere o elemento especificado no início da deque. |
| E | remove | Retorna e remove o primeiro elemento da deque. |
| E | removeFirst | Retorna e remove o primeiro elemento da deque. |
| E | removeLast | Retorna e remove o último elemento da deque. |
| int | size | Retorna o número de elementos na deque. |

## Métodos da Classe LinkedList

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Adiciona o elemento especificado como o último elemento nesta lista (equivalente a addLast(E e)). |
| void | addFirst(E e) | Insere o elemento especificado no início desta lista (equivalente a offerFirst(E e)). |
| void | addLast(E e) | Insere o elemento especificado no final desta lista (equivalente a offerLast(E e)). |
| void | clear() | Remove todos os elementos desta lista. |
| Object | clone() | Retorna um clone desta lista. |
| boolean | contains(Object o) | Retorna verdadeiro se esta lista contiver o elemento especificado. |
| Iterator<E> | descendingIterator() | Retorna um iterator sobre os elementos em ordem descendente. |
| E | element() | Retorna o primeiro elemento nesta lista (sem removê-lo). |
| E | getFirst() | Retorna o primeiro elemento nesta lista (sem removê-lo, equivalente a element()). |
| E | getLast() | Retorna o último elemento nesta lista (sem removê-lo). |
| boolean | isEmpty() | Retorna verdadeiro se esta lista não contiver nenhum elemento. |
| Iterator<E> | iterator() | Retorna um iterator sobre os elementos nesta lista (em ordem da cabeça à cauda). |
| boolean | offer(E e) | Adiciona o elemento especificado como o último elemento nesta lista (equivalente a add(E e)). |
| boolean | offerFirst(E e) | Insere o elemento especificado no início desta lista. |
| boolean | offerLast(E e) | Insere o elemento especificado no final desta lista. |
| E | peek() | Retorna o primeiro elemento nesta lista (sem removê-lo, ou retorna null se esta lista estiver vazia). |
| E | peekFirst() | Retorna o primeiro elemento nesta lista (sem removê-lo, ou retorna null se esta lista estiver vazia, equivalente a peek()). |
| E | peekLast() | Retorna o último elemento nesta lista (sem removê-lo, ou retorna null se esta lista estiver vazia). |
| E | poll() | Remove e retorna o primeiro elemento desta lista, ou retorna null se esta lista estiver vazia. |
| E | pollFirst() | Remove e retorna o primeiro elemento desta lista, ou retorna null se esta lista estiver vazia (equivalente a poll()). |
| E | pollLast() | Remove e retorna o último elemento desta lista, ou retorna null se esta lista estiver vazia. |
| E | pop() | Remove e retorna o primeiro elemento desta lista (equivalente a removeFirst()). |
| void | push(E e) | Insere o elemento especificado na frente desta lista (equivalente a addFirst(E e)). |
| E | remove() | Remove e retorna o primeiro elemento desta lista (equivalente a removeFirst()). |
| boolean | remove(Object o) | Remove a primeira ocorrência do elemento especificado desta lista, se presente. |
| E | removeFirst() | Remove e retorna o primeiro elemento desta lista. |
| boolean | removeFirstOccurrence(Object o) | Remove a primeira ocorrência do elemento especificado nesta lista, se presente. |
| E | removeLast() | Remove e retorna o último elemento desta lista. |
| boolean | removeLastOccurrence(Object o) | Remove a última ocorrência do elemento especificado nesta lista, se presente. |
| int | size() | Retorna o número de elementos |

## Métodos da Interface BlockingDeque

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | addFirst(E e) | Insere o elemento especificado no início desta deque, se possível, imediatamente retornando true, lançando uma IllegalStateException se a capacidade da deque estiver esgotada. |
| boolean | addLast(E e) | Insere o elemento especificado no final desta deque, se possível, imediatamente retornando true, lançando uma IllegalStateException se a capacidade da deque estiver esgotada. |
| void | clear() | Remove todos os elementos desta deque. |
| Object | clone() | Retorna uma cópia desta deque. |
| boolean | contains(Object o) | Retorna true se esta deque contiver o elemento especificado. |
| Iterator | descendingIterator() | Retorna um iterator sobre os elementos na ordem reversa. |
| E | element() | Recupera, mas não remove, o primeiro elemento desta deque. |
| E | getFirst() | Recupera, mas não remove, o primeiro elemento desta deque. |
| E | getLast() | Recupera, mas não remove, o último elemento desta deque. |
| boolean | offer(E e) | Insere o elemento especificado no final desta deque, se possível. |
| boolean | offerFirst(E e) | Insere o elemento especificado no início desta deque, se possível. |
| boolean | offerLast(E e) | Insere o elemento especificado no final desta deque, se possível. |
| E | peek() | Recupera, mas não remove, o primeiro elemento desta deque. |
| E | peekFirst() | Recupera, mas não remove, o primeiro elemento desta deque. |
| E | peekLast() | Recupera, mas não remove, o último elemento desta deque. |
| E | poll() | Recupera e remove o primeiro elemento desta deque, ou retorna null se esta deque estiver vazia. |
| E | pollFirst() | Recupera e remove o primeiro elemento desta deque, ou retorna null se esta deque estiver vazia. |
| E | pollLast() | Recupera e remove o último elemento desta deque, ou retorna null se esta deque estiver vazia. |
| void | put(E e) | Insere o elemento especificado no final desta deque, aguardando se necessário até que a capacidade permita. |
| int | remainingCapacity() | Retorna a quantidade de elementos adicionais que esta deque pode acomodar. |
| E | remove() | Recupera e remove o primeiro elemento desta deque. |
| boolean | remove(Object o) | Remove a primeira ocorrência do elemento especificado nesta deque. |
| E | removeFirst() | Recupera e remove o primeiro elemento desta deque. |
| boolean | removeFirstOccurrence(Object o) | Remove a primeira ocorrência do elemento especificado desta deque. |
| E | removeLast() | Recupera e remove o último elemento desta deque. |
| boolean | removeLastOccurrence(Object o) | Remove a última ocorrência do elemento especificado desta deque. |
| int | size() | Retorna o número de elementos nesta deque. |
| E | take() | Recupera e remove o primeiro elemento desta deque, aguardando se necessário até que um elemento esteja disponível. |
| Object[] | toArray() | Retorna um array contendo todos os elementos desta deque na ordem em que são |

## Métodos da Classe LinkedBlockingDeque

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Insere o elemento especificado no final desta deque |
| void | addFirst(E e) | Insere o elemento especificado no início desta deque |
| void | addLast(E e) | Insere o elemento especificado no final desta deque |
| void | clear() | Remove todos os elementos desta deque |
| Object | clone() | Retorna uma cópia superficial desta deque |
| boolean | contains(Object o) | Retorna true se esta deque contém o elemento |
| Iterator<E> | descendingIterator() | Retorna um iterador sobre os elementos desta deque |
| E | element() | Retorna o primeiro elemento desta deque |
| E | getFirst() | Retorna o primeiro elemento desta deque |
| E | getLast() | Retorna o último elemento desta deque |
| boolean | isEmpty() | Retorna true se esta deque estiver vazia |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos desta deque |
| boolean | offer(E e) | Insere o elemento especificado no final desta deque |
| boolean | offerFirst(E e) | Insere o elemento especificado no início desta deque |
| boolean | offerLast(E e) | Insere o elemento especificado no final desta deque |
| E | peek() | Retorna o primeiro elemento desta deque, ou null se esta deque estiver vazia |
| E | peekFirst() | Retorna o primeiro elemento desta deque, ou null se esta deque estiver vazia |
| E | peekLast() | Retorna o último elemento desta deque, ou null se esta deque estiver vazia |
| E | poll() | Retorna e remove o primeiro elemento desta deque, ou null se esta deque estiver vazia |
| E | pollFirst() | Retorna e remove o primeiro elemento desta deque, ou null se esta deque estiver vazia |
| E | pollLast() | Retorna e remove o último elemento desta deque, ou null se esta deque estiver vazia |
| void | put(E e) | Insere o elemento especificado no final desta deque, aguardando se necessário para fazer isso |
| void | putFirst(E e) | Insere o elemento especificado no início desta deque, aguardando se necessário para fazer isso |
| void | putLast(E e) | Insere o elemento especificado no final desta deque, aguardando se necessário para fazer isso |
| int | remainingCapacity() | Retorna a capacidade restante desta deque |
| E | remove() | Retorna e remove o primeiro elemento desta deque |
| boolean | remove(Object o) | Remove a primeira ocorrência do elemento especificado desta deque |
| E | removeFirst() | Retorna e remove o primeiro elemento desta deque |
| boolean | removeFirstOccurrence(Object o) | Remove a primeira ocorrência do elemento especificado desta deque a partir do início |
| E | removeLast() | Retorna e remove o último elemento desta deque |
| boolean | removeLastOccurrence(Object o) | Remove a primeira ocorrência do elemento especificado desta deque a partir do final |
| int | size() | Retorna o número de elementos nesta deque. |
| E[] | toArray() | Retorna uma array contendo todos os elementos desta deque na ordem em que estão. |

## Métodos da Interface Set

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(boolean e) | Adiciona o elemento especificado ao conjunto, se ele ainda não estiver presente. |
| boolean | add(E e) | Adiciona o elemento especificado ao conjunto, se ele ainda não estiver presente. |
| boolean | addAll(Collection<? extends E> c) | Adiciona todos os elementos na coleção especificada ao conjunto, se eles ainda não estiverem presentes. |
| void | clear() | Remove todos os elementos deste conjunto (o conjunto ficará vazio). |
| Object | clone() | Retorna uma cópia rasa deste conjunto: os elementos em si não são clonados. |
| boolean | contains(boolean o) | Retorna verdadeiro se este conjunto contiver o elemento especificado. |
| boolean | contains(Object o) | Retorna verdadeiro se este conjunto contiver o elemento especificado. |
| boolean | containsAll(Collection<?> c) | Retorna verdadeiro se este conjunto contiver todos os elementos na coleção especificada. |
| boolean | equals(Object o) | Compara o objeto especificado com este conjunto para igualdade. |
| int | hashCode() | Retorna o código hash para este conjunto. |
| boolean | isEmpty() | Retorna verdadeiro se este conjunto não contiver elementos. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos contidos neste conjunto. |
| boolean | remove(boolean o) | Remove o elemento especificado deste conjunto, se ele estiver presente. |
| boolean | remove(Object o) | Remove o elemento especificado deste conjunto, se ele estiver presente. |
| boolean | removeAll(Collection<?> c) | Remove todos os elementos deste conjunto que também estão contidos na coleção especificada. |
| boolean | retainAll(Collection<?> c) | Retém apenas os elementos deste conjunto que estão contidos na coleção especificada. |
| int | size() | Retorna o número de elementos neste conjunto (sua cardinalidade). |
| int | size() | Retorna o número de elementos neste conjunto (sua cardinalidade). |
| Object[] | toArray() | Retorna um array contendo todos os elementos neste conjunto. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos neste conjunto; o tipo de tempo de execução do array retornado é o tipo do array especificado. |

## Métodos da Interface SortedSet

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| Comparator<? super E> | comparator() | Retorna o comparador usado para ordenar os elementos do conjunto, ou null se o conjunto usa a ordem natural dos elementos |
| E | first() | Retorna o primeiro (menor) elemento do conjunto |
| SortedSet<E> | headSet(E toElement) | Retorna um subconjunto do conjunto contendo todos os elementos menores que o elemento especificado |
| E | last() | Retorna o último (maior) elemento do conjunto |
| SortedSet<E> | subSet(E fromElement, E toElement) | Retorna um subconjunto do conjunto contendo todos os elementos maiores ou iguais ao elemento de início e menores que o elemento final especificado |
| SortedSet<E> | tailSet(E fromElement) | Retorna um subconjunto do conjunto contendo todos os elementos maiores ou iguais ao elemento especificado |

## Métodos da Interface NavigableSet

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Adiciona o elemento especificado ao conjunto |
| boolean | addAll(Collection<? extends E> c) | Adiciona todos os elementos da coleção especificada a este conjunto |
| void | clear() | Remove todos os elementos deste conjunto |
| Comparator<? super E> | comparator() | Retorna o comparador usado para ordenar os elementos neste conjunto |
| boolean | contains(Object o) | Retorna true se este conjunto contém o elemento especificado |
| Iterator<E> | descendingIterator() | Retorna um iterador sobre os elementos deste conjunto em ordem decrescente |
| NavigableSet<E> | descendingSet() | Retorna um conjunto com os elementos deste conjunto em ordem decrescente |
| E | first() | Retorna o primeiro (menor) elemento deste conjunto |
| E | floor(E e) | Retorna o maior elemento deste conjunto menor ou igual ao elemento especificado |
| SortedSet<E> | headSet(E toElement) | Retorna um conjunto com todos os elementos deste conjunto menores que o elemento especificado |
| NavigableSet<E> | headSet(E toElement, boolean inclusive) | Retorna um conjunto com todos os elementos deste conjunto menores que o elemento especificado, opcionalmente incluindo o elemento especificado |
| E | higher(E e) | Retorna o menor elemento deste conjunto estritamente maior que o elemento especificado |
| boolean | isEmpty() | Retorna true se este conjunto não contiver nenhum elemento |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos deste conjunto em ordem ascendente |
| E | last() | Retorna o último (maior) elemento deste conjunto |
| E | lower(E e) | Retorna o maior elemento deste conjunto estritamente menor que o elemento especificado |
| E | pollFirst() | Retorna e remove o primeiro (menor) elemento deste conjunto, ou retorna null se este conjunto estiver vazio |
| E | pollLast() | Retorna e remove o último (maior) elemento deste conjunto, ou retorna null se este conjunto estiver vazio |
| boolean | remove(Object o) | Remove o elemento especificado deste conjunto, se ele estiver presente |
| int | size() | Retorna o número de elementos neste conjunto |
| NavigableSet<E> | subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) | Retorna um conjunto com todos os elementos deste conjunto maiores ou iguais que o elemento especificado, e menores que o elemento especificado, opcionalmente incluindo os elementos especificados |
| SortedSet<E> | subSet(E fromElement, E toElement) | Retorna um conjunto com todos os elementos deste conjunto maiores ou iguais que o elemento especificado, e menores que o elemento especificado |
| SortedSet<E> | tailSet(E fromElement) | Retorna um conjunto com todos os elementos deste conjunto maiores ou iguais que o elemento especificado |
| NavigableSet<E> | tailSet(E fromElement, boolean inclusive) | Retorna um conjunto com todos os elementos deste conjunto maiores ou iguais que o elemento especificado, opcionalmente incluindo o elemento especificado |

## Métodos da Classe TreeSet

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Adiciona o elemento especificado a este conjunto se ele ainda não estiver presente. |
| boolean | addAll(Collection<? extends E> c) | Adiciona todos os elementos na coleção especificada a este conjunto, se eles ainda não estiverem presentes. |
| void | clear() | Remove todos os elementos deste conjunto. |
| Object | clone() | Retorna uma cópia rasa deste conjunto: os elementos em si não são clonados. |
| Comparator<? super E> | comparator() | Retorna o comparador usado para ordenar este conjunto, ou nulo se ele usa a ordem natural dos elementos. |
| boolean | contains(Object o) | Retorna verdadeiro se este conjunto contiver o elemento especificado. |
| Iterator<E> | descendingIterator() | Retorna um iterador sobre os elementos neste conjunto em ordem decrescente. |
| NavigableSet<E> | descendingSet() | Retorna uma visão reversa deste conjunto. |
| E | first() | Retorna o primeiro (menor) elemento atualmente neste conjunto. |
| E | floor(E e) | Retorna o maior elemento neste conjunto menor ou igual ao elemento especificado, ou nulo se não houver tal elemento. |
| SortedSet<E> | headSet(E toElement) | Retorna uma visão deste conjunto cujos elementos são estritamente menores que toElement. |
| NavigableSet<E> | headSet(E toElement, boolean inclusive) | Retorna uma visão deste conjunto cujos elementos são estritamente menores que toElement. |
| E | higher(E e) | Retorna o menor elemento neste conjunto estritamente maior que o elemento especificado, ou nulo se não houver tal elemento. |
| boolean | isEmpty() | Retorna verdadeiro se este conjunto não contiver elementos. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos neste conjunto em ordem crescente. |
| E | last() | Retorna o último (maior) elemento atualmente neste conjunto. |
| E | lower(E e) | Retorna o maior elemento neste conjunto estritamente menor que o elemento especificado, ou nulo se não houver tal elemento. |
| E | pollFirst() | Remove e retorna o primeiro (menor) elemento deste conjunto. |
| E | pollLast() | Remove e retorna o último (maior) elemento deste conjunto. |
| boolean | remove(Object o) | Remove o elemento especificado deste conjunto, se ele estiver presente. |
| int | size() | Retorna o número de elementos neste conjunto (sua cardinalidade). |
| NavigableSet<E> | subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) | Retorna uma visão deste conjunto cujos elementos vão do elemento fromElement (inclusive) ao elemento toElement (exclusive). |
| SortedSet<E> | subSet(E fromElement, E toElement) | Retorna uma visão deste conjunto cujos elementos vão do elemento fromElement (inclusive) ao elemento toElement (exclusive). |
| SortedSet<E> | tailSet(E fromElement) | Retorna uma visão deste conjunto cujos elementos são maiores ou iguais a fromElement. |
| NavigableSet<E> | tailSet(E fromElement, boolean inclusive) | Retorna uma visão deste conjunto cujos elementos são maiores ou iguais a fromElement. |
| Object[] | toArray() | Retorna um array contendo todos os elementos neste conjunto. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos neste conjunto; o tipo de tempo de execução do array retornado é o tipo do array especificado. |

## Métodos da Classe ConcurrentSkipListSet

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Adiciona o elemento especificado a este conjunto se ele ainda não estiver presente. |
| boolean | addAll(Collection<? extends E> c) | Adiciona todos os elementos na coleção especificada a este conjunto, se eles ainda não estiverem presentes. |
| void | clear() | Remove todos os elementos deste conjunto. |
| ConcurrentSkipListSet<E> | clone() | Retorna uma cópia rasa deste conjunto: os elementos em si não são clonados. |
| boolean | contains(Object o) | Retorna verdadeiro se este conjunto contiver o elemento especificado. |
| Iterator<E> | descendingIterator() | Retorna um iterador sobre os elementos neste conjunto em ordem decrescente. |
| NavigableSet<E> | descendingSet() | Retorna uma visão reversa deste conjunto. |
| E | first() | Retorna o primeiro (menor) elemento atualmente neste conjunto. |
| E | floor(E e) | Retorna o maior elemento neste conjunto menor ou igual ao elemento especificado, ou nulo se não houver tal elemento. |
| SortedSet<E> | headSet(E toElement) | Retorna uma visão deste conjunto cujos elementos são estritamente menores que toElement. |
| NavigableSet<E> | headSet(E toElement, boolean inclusive) | Retorna uma visão deste conjunto cujos elementos são estritamente menores que toElement. |
| E | higher(E e) | Retorna o menor elemento neste conjunto estritamente maior que o elemento especificado, ou nulo se não houver tal elemento. |
| boolean | isEmpty() | Retorna verdadeiro se este conjunto não contiver elementos. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos neste conjunto em ordem crescente. |
| E | last() | Retorna o último (maior) elemento atualmente neste conjunto. |
| E | lower(E e) | Retorna o maior elemento neste conjunto estritamente menor que o elemento especificado, ou nulo se não houver tal elemento. |
| E | pollFirst() | Remove e retorna o primeiro (menor) elemento deste conjunto. |
| E | pollLast() | Remove e retorna o último (maior) elemento deste conjunto. |
| boolean | remove(Object o) | Remove o elemento especificado deste conjunto, se ele estiver presente. |
| int | size() | Retorna o número de elementos neste conjunto (sua cardinalidade). |
| NavigableSet<E> | subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) | Retorna uma visão deste conjunto cujos elementos vão do elemento fromElement (inclusive) ao elemento toElement (exclusive). |
| SortedSet<E> | subSet(E fromElement, E toElement) | Retorna uma visão deste conjunto cujos elementos vão do elemento fromElement (inclusive) ao elemento toElement (exclusive). |
| SortedSet<E> | tailSet(E fromElement) | Retorna uma visão deste conjunto cujos elementos são maiores ou iguais a fromElement. |
| NavigableSet<E> | tailSet(E fromElement, boolean inclusive) | Retorna uma visão deste conjunto cujos elementos são maiores ou iguais a fromElement. |
| Object[] | toArray() | Retorna um array contendo todos os elementos neste conjunto. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos neste conjunto; o tipo de tempo de execução do array retornado é o tipo do array especificado. |

## Métodos da Classe HashSet

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Adiciona o elemento especificado a este conjunto se ele ainda não estiver presente. |
| boolean | addAll(Collection<? extends E> c) | Adiciona todos os elementos na coleção especificada a este conjunto, se eles ainda não estiverem presentes. |
| void | clear() | Remove todos os elementos deste conjunto. |
| Object | clone() | Retorna uma cópia rasa deste conjunto: os elementos em si não são clonados. |
| boolean | contains(Object o) | Retorna verdadeiro se este conjunto contiver o elemento especificado. |
| boolean | containsAll(Collection<?> c) | Retorna verdadeiro se este conjunto contiver todos os elementos na coleção especificada. |
| boolean | equals(Object o) | Compara o objeto especificado com este conjunto para igualdade. |
| int | hashCode() | Retorna o código de hash para este conjunto. |
| boolean | isEmpty() | Retorna verdadeiro se este conjunto não contiver elementos. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos neste conjunto. |
| boolean | remove(Object o) | Remove o elemento especificado deste conjunto, se ele estiver presente. |
| boolean | removeAll(Collection<?> c) | Remove todos os elementos deste conjunto que também estão contidos na coleção especificada. |
| boolean | retainAll(Collection<?> c) | Remove todos os elementos deste conjunto que não estão contidos na coleção especificada. |
| int | size() | Retorna o número de elementos neste conjunto (sua cardinalidade). |
| Object[] | toArray() | Retorna um array contendo todos os elementos neste conjunto. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos neste conjunto; o tipo de tempo de execução do array retornado é o do array especificado. |

## Métodos da Classe LinkedHashSet

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Adiciona o elemento especificado a este conjunto se ele ainda não estiver presente. |
| boolean | addAll(Collection<? extends E> c) | Adiciona todos os elementos na coleção especificada a este conjunto, se eles ainda não estiverem presentes. |
| void | clear() | Remove todos os elementos deste conjunto. |
| Object | clone() | Retorna uma cópia rasa deste conjunto: os elementos em si não são clonados. |
| boolean | contains(Object o) | Retorna verdadeiro se este conjunto contiver o elemento especificado. |
| boolean | containsAll(Collection<?> c) | Retorna verdadeiro se este conjunto contiver todos os elementos na coleção especificada. |
| boolean | equals(Object o) | Compara o objeto especificado com este conjunto para igualdade. |
| int | hashCode() | Retorna o código de hash para este conjunto. |
| boolean | isEmpty() | Retorna verdadeiro se este conjunto não contiver elementos. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos neste conjunto na ordem em que foram inseridos. |
| boolean | remove(Object o) | Remove o elemento especificado deste conjunto, se ele estiver presente. |
| boolean | removeAll(Collection<?> c) | Remove todos os elementos deste conjunto que também estão contidos na coleção especificada. |
| boolean | retainAll(Collection<?> c) | Remove todos os elementos deste conjunto que não estão contidos na coleção especificada. |
| int | size() | Retorna o número de elementos neste conjunto (sua cardinalidade). |
| Object[] | toArray() | Retorna um array contendo todos os elementos neste conjunto na ordem em que foram inseridos. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos neste conjunto na ordem em que foram inseridos; o tipo de tempo de execução do array retornado é o do array especificado. |

## Métodos da Classe CopyOnWriteArraySet

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| boolean | add(E e) | Adiciona o elemento especificado a este conjunto se ele ainda não estiver presente. |
| boolean | addAll(Collection<? extends E> c) | Adiciona todos os elementos na coleção especificada a este conjunto, se eles ainda não estiverem presentes. |
| void | clear() | Remove todos os elementos deste conjunto. |
| Object | clone() | Retorna uma cópia rasa deste conjunto: os elementos em si não são clonados. |
| boolean | contains(Object o) | Retorna verdadeiro se este conjunto contiver o elemento especificado. |
| boolean | containsAll(Collection<?> c) | Retorna verdadeiro se este conjunto contiver todos os elementos na coleção especificada. |
| boolean | equals(Object o) | Compara o objeto especificado com este conjunto para igualdade. |
| int | hashCode() | Retorna o código de hash para este conjunto. |
| boolean | isEmpty() | Retorna verdadeiro se este conjunto não contiver elementos. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos neste conjunto na ordem em que foram inseridos. |
| boolean | remove(Object o) | Remove o elemento especificado deste conjunto, se ele estiver presente. |
| boolean | removeAll(Collection<?> c) | Remove todos os elementos deste conjunto que também estão contidos na coleção especificada. |
| boolean | retainAll(Collection<?> c) | Remove todos os elementos deste conjunto que não estão contidos na coleção especificada. |
| int | size() | Retorna o número de elementos neste conjunto (sua cardinalidade). |
| Object[] | toArray() | Retorna um array contendo todos os elementos neste conjunto na ordem em que foram inseridos. |
| <T> T[] | toArray(T[] a) | Retorna um array contendo todos os elementos neste conjunto na ordem em que foram inseridos; o tipo de tempo de execução do array retornado é o do array especificado. |

## Métodos da Classe EnumSet

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | add(E e) | Adiciona o elemento especificado a este conjunto. |
| static <E extends Enum<E>> EnumSet<E> | allOf(Class<E> elementType) | Retorna um conjunto que é uma visualização "especializada" do conjunto de todos os valores possíveis do tipo de enumeração especificado. |
| void | clear() | Remove todos os elementos deste conjunto. |
| Object | clone() | Retorna uma cópia deste conjunto. |
| static <E extends Enum<E>> EnumSet<E> | complementOf(EnumSet<E> s) | Retorna um conjunto cujos elementos são complementares aos elementos no conjunto especificado. |
| boolean | contains(Object o) | Retorna verdadeiro se este conjunto contiver o elemento especificado. |
| static <E extends Enum<E>> EnumSet<E> | copyOf(Collection<E> c) | Retorna um conjunto cujos elementos são os mesmos que os elementos no conjunto especificado. |
| static <E extends Enum<E>> EnumSet<E> | copyOf(EnumSet<E> s) | Retorna um conjunto cujos elementos são os mesmos que os elementos no conjunto especificado. |
| void | forEach(Consumer<? super E> action) | Executa a ação especificada em cada elemento deste conjunto. |
| Iterator<E> | iterator() | Retorna um iterador sobre os elementos neste conjunto. |
| static <E extends Enum<E>> EnumSet<E> | noneOf(Class<E> elementType) | Retorna um conjunto vazio com o tipo de enumeração especificado. |
| static <E extends Enum<E>> EnumSet<E> | of(E e) | Retorna um conjunto contendo apenas o elemento especificado. |
| static <E extends Enum<E>> EnumSet<E> | of(E e1, E e2) | Retorna um conjunto contendo os dois elementos especificados. |
| static <E extends Enum<E>> EnumSet<E> | of(E e1, E e2, E e3) | Retorna um conjunto contendo os três elementos especificados. |
| static <E extends Enum<E>> EnumSet<E> | of(E e1, E e2, E e3, E e4) | Retorna um conjunto contendo os quatro elementos especificados. |
| static <E extends Enum<E>> EnumSet<E> | of(E e1, E e2, E e3, E e4, E e5) | Retorna um conjunto contendo os cinco elementos especificados. |
| static <E extends Enum<E>> EnumSet<E> | of(E e1, E e2, E e3, E e4, E e5, E e6, E... elements) | Retorna um conjunto contendo os elementos especificados. |
| void | remove(Object o) | Remove o elemento especificado deste conjunto, se estiver presente. |
| int | size() | Retorna o número de elementos neste conjunto. |

## Métodos da Interface Map

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | clear() | É usado para redefinir o mapa. |
| V | compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) | Ele é usado para calcular um mapeamento para a chave especificada e seu valor mapeado atual (ou nulo se não houver mapeamento atual). |
| V | computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction) | Ele é usado para calcular seu valor usando a função de mapeamento fornecida, se a chave especificada ainda não estiver associada a um valor (ou mapeada para nulo) e a insere nesse mapa, a menos que seja nula. |
| V | computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) | Ele é usado para calcular um novo mapeamento dado a chave e seu valor mapeado atual se o valor para a chave especificada estiver presente e não nulo. |
| boolean | containsKey(Object key) | Este método retorna verdadeiro se alguma chave igual à chave existir dentro do mapa, senão retorna falso. |
| boolean | containsValue(Object value) | Este método retorna verdadeiro se algum valor igual ao valor existir dentro do mapa, senão retorna falso. |
| boolean | equals(Object o) | É usado para comparar o Objeto especificado com o Mapa. |
| void | forEach(BiConsumer<? super K,? super V> action) | Ele executa a ação dada para cada entrada no mapa até que todas as entradas tenham sido processadas ou a ação lance uma exceção. |
| V | get(Object key) | Este método retorna o objeto que contém o valor associado à chave. |
| V | getOrDefault(Object key, V defaultValue) | Ele retorna o valor para o qual a chave especificada é mapeada ou defaultValue se o mapa não contiver mapeamento para a chave. |
| int | hashCode() | Ele retorna o valor do código hash para o mapa |
| boolean | isEmpty() | Este método retorna true se o mapa estiver vazio; retorna falso se contiver pelo menos uma chave. |
| V | merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction) | Se a chave especificada ainda não estiver associada a um valor ou estiver associada a nulo, associe-a ao valor não nulo fornecido. |
| V | put(Object key, Object value) | É usado para inserir uma entrada no mapa. |
| void | putAll(Map map) | É usado para inserir o mapa especificado no mapa. |
| void | putIfAbsent(K key, V value) | Ele insere o valor especificado com a chave especificada no mapa somente se ainda não estiver especificado. |
| void | remove(Object key) | Ele é usado para excluir uma entrada para a chave especificada. |
| boolean | remove(Object key, Object value) | Ele remove os valores especificados com as chaves especificadas associadas do mapa. |
| boolean | replace(K key, V oldValue, V newValue) | Ele substitui o valor antigo pelo novo valor para uma chave especificada. |
| V | replace(K key, V value) | Ele substitui o valor especificado por uma chave especificada. |
| void | replaceAll(BiFunction<? super K,? super V,? extends V> function) | Ele substitui o valor de cada entrada pelo resultado da invocação da função fornecida nessa entrada até que todas as entradas tenham sido processadas ou a função lance uma exceção. |
| Set<K> | Set keySet() | Ele retorna à visualização Set contendo todas as chaves. |
| int | size() | Este método retorna o número de entradas no mapa. |
| Collection<V> | values() | Ele retorna uma visualização de coleção dos valores contidos no mapa. |

## Métodos da Interface ConcurrentMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | clear() | Remove todos os mapeamentos deste mapa. |
| V | compute(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction) | Tentar computar um novo mapeamento para a chave e seu valor atual no mapa. |
| V | computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction) | Computa um valor se a chave especificada não estiver presente ou estiver mapeada como nula. |
| V | computeIfPresent(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction) | Computa um novo mapeamento para a chave especificada somente se ela atualmente estiver mapeada para algum valor. |
| void | forEach(BiConsumer<? super K, ? super V> action) | Execute a ação especificada para cada entrada no mapa até que todas as entradas tenham sido processadas ou a ação tenha lançado uma exceção. |
| V | get(Object key) | Retorna o valor ao qual a chave especificada está mapeada ou nulo se este mapa não contiver um mapeamento para a chave especificada. |
| V | getOrDefault(Object key, V defaultValue) | Retorna o valor ao qual a chave especificada está mapeada ou o valor padrão fornecido se este mapa não contiver um mapeamento para a chave especificada. |
| V | merge(K key, V value, BiFunction<? super V, ? super V, ? extends V> remappingFunction) | Combina o valor fornecido com o valor atualmente associado à chave especificada usando a função especificada e armazena o resultado no mapa. |
| V | put(K key, V value) | Associa o valor especificado com a chave especificada nesse mapa (operando atomicamente). |
| V | putIfAbsent(K key, V value) | Associa o valor especificado com a chave especificada nesse mapa, somente se nenhum valor anterior estiver associado à chave especificada. |
| boolean | remove(Object key) | Remove o mapeamento para a chave especificada deste mapa, se presente. |
| boolean | remove(Object key, Object value) | Remove o par chave-valor associado à chave especificada somente se estiver atualmente mapeado para o valor especificado. |
| boolean | remove(Object key, Object value) | Remove o par chave-valor associado à chave especificada somente se estiver atualmente mapeado para o valor especificado. |
| boolean | replace(K key, V oldValue, V newValue) | Substitui a entrada para uma chave somente se atualmente mapeada para um determinado valor. |
| boolean | replace(K key, V oldValue, V newValue) | Substitui a entrada para uma chave somente se atualmente mapeada para um determinado valor. |
| boolean | replace(K key, V value) | Substitui a entrada para uma chave somente se atualmente mapeada para algum valor. |
| void | replaceAll(BiFunction<? super K, ? super V, ? extends V> function) | Substitui cada valor deste mapa com o resultado da aplicação da função dada a cada chave-valor. |

## Métodos da Interface SortedMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| Comparator<? super K> | comparator() | Retorna o comparador usado para ordenar as chaves deste mapa ordenado ou nulo se este mapa usa a ordem natural das chaves. |
| Set<Map.Entry<K, V>> | entrySet() | Retorna um conjunto de todas as entradas deste mapa. |
| V | firstEntry() | Retorna a primeira entrada neste mapa ou nulo se o mapa estiver vazio. |
| K | firstKey() | Retorna a primeira (menor) chave neste mapa ordenado. |
| V | get(Object key) | Retorna o valor ao qual a chave especificada está mapeada ou nulo se este mapa não contiver um mapeamento para a chave especificada. |
| SortedMap<K, V> | headMap(K toKey) | Retorna uma visão de cabeça de mapa do mapa deste mapa, cujas chaves são estritamente menores que toKey. |
| SortedMap<K, V> | headMap(K toKey) | Retorna uma visão de cabeça de mapa do mapa deste mapa, cujas chaves são estritamente menores que toKey. |
| Set<K> | keySet() | Retorna um conjunto de todas as chaves deste mapa. |
| V | lastEntry() | Retorna a última entrada neste mapa ou nulo se o mapa estiver vazio. |
| K | lastKey() | Retorna a última (maior) chave neste mapa ordenado. |
| V | pollFirstEntry() | Remove e retorna uma entrada mapeada para a primeira (menor) chave neste mapa ou nulo se o mapa estiver vazio. |
| V | pollLastEntry() | Remove e retorna uma entrada mapeada para a última (maior) chave neste mapa ou nulo se o mapa estiver vazio. |
| V | put(K key, V value) | Associa o valor especificado com a chave especificada neste mapa (operando atomicamente). |
| V | remove(Object key) | Remove o mapeamento para a chave especificada deste mapa, se presente. |
| int | size() | Retorna o número de mapeamentos |
| SortedMap<K, V> | subMap(K fromKey, K toKey) | Retorna uma visão de mapa deste mapa, cujas chaves variam de fromKey, inclusive, a toKey, exclusivo. |
| SortedMap<K, V> | subMap(K fromKey, K toKey) | Retorna uma visão de mapa deste mapa, cujas chaves variam de fromKey, inclusive, a toKey, exclusivo. |
| SortedMap<K, V> | tailMap(K fromKey) | Retorna uma visão de cauda de mapa do mapa deste mapa, cujas chaves são maiores ou iguais a fromKey. |
| SortedMap<K, V> | tailMap(K fromKey) | Retorna uma visão de cauda de mapa do mapa deste mapa, cujas chaves são maiores ou iguais a fromKey. |
| Collection<V> | values() | Retorna uma coleção de todos os valores deste mapa. |

## Métodos da Interface NavigableMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| Comparator<? super K> | comparator() | Retorna o comparador usado para ordenar as chaves deste mapa ordenado ou nulo se este mapa usa a ordem natural das chaves. |
| Set<Map.Entry<K, V>> | entrySet() | Retorna um conjunto de todas as entradas deste mapa. |
| V | firstEntry() | Retorna a primeira entrada neste mapa ou nulo se o mapa estiver vazio. |
| K | firstKey() | Retorna a primeira (menor) chave neste mapa ordenado. |
| V | get(Object key) | Retorna o valor ao qual a chave especificada está mapeada ou nulo se este mapa não contiver um mapeamento para a chave especificada. |
| SortedMap<K, V> | headMap(K toKey) | Retorna uma visão de cabeça de mapa do mapa deste mapa, cujas chaves são estritamente menores que toKey. |
| SortedMap<K, V> | headMap(K toKey) | Retorna uma visão de cabeça de mapa do mapa deste mapa, cujas chaves são estritamente menores que toKey. |
| Set<K> | keySet() | Retorna um conjunto de todas as chaves deste mapa. |
| V | lastEntry() | Retorna a última entrada neste mapa ou nulo se o mapa estiver vazio. |
| K | lastKey() | Retorna a última (maior) chave neste mapa ordenado. |
| V | pollFirstEntry() | Remove e retorna uma entrada mapeada para a primeira (menor) chave neste mapa ou nulo se o mapa estiver vazio. |
| V | pollLastEntry() | Remove e retorna uma entrada mapeada para a última (maior) chave neste mapa ou nulo se o mapa estiver vazio. |
| V | put(K key, V value) | Associa o valor especificado com a chave especificada neste mapa (operando atomicamente). |
| V | remove(Object key) | Remove o mapeamento para a chave especificada deste mapa, se presente. |
| int | size() | Retorna o número de mapeamentos |
| SortedMap<K, V> | subMap(K fromKey, K toKey) | Retorna uma visão de mapa deste mapa, cujas chaves variam de fromKey, inclusive, a toKey, exclusivo. |
| SortedMap<K, V> | subMap(K fromKey, K toKey) | Retorna uma visão de mapa deste mapa, cujas chaves variam de fromKey, inclusive, a toKey, exclusivo. |
| SortedMap<K, V> | tailMap(K fromKey) | Retorna uma visão de cauda de mapa do mapa deste mapa, cujas chaves são maiores ou iguais a fromKey. |
| SortedMap<K, V> | tailMap(K fromKey) | Retorna uma visão de cauda de mapa do mapa deste mapa, cujas chaves são maiores ou iguais a fromKey. |
| Collection<V> | values() | Retorna uma coleção de todos os valores deste mapa. |

## Métodos da Interface ConcurrentNavigableMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| Comparator<? super K> | comparator() | Retorna o comparador usado para ordenar as chaves deste mapa ordenado ou nulo se este mapa usa a ordem natural das chaves. |
| Set<Map.Entry<K, V>> | entrySet() | Retorna um conjunto de todas as entradas deste mapa. |
| V | firstEntry() | Retorna a primeira entrada neste mapa ou nulo se o mapa estiver vazio. |
| K | firstKey() | Retorna a primeira (menor) chave neste mapa ordenado. |
| V | get(Object key) | Retorna o valor ao qual a chave especificada está mapeada ou nulo se este mapa não contiver um mapeamento para a chave especificada. |
| SortedMap<K, V> | headMap(K toKey) | Retorna uma visão de cabeça de mapa do mapa deste mapa, cujas chaves são estritamente menores que toKey. |
| SortedMap<K, V> | headMap(K toKey) | Retorna uma visão de cabeça de mapa do mapa deste mapa, cujas chaves são estritamente menores que toKey. |
| Set<K> | keySet() | Retorna um conjunto de todas as chaves deste mapa. |
| V | lastEntry() | Retorna a última entrada neste mapa ou nulo se o mapa estiver vazio. |
| K | lastKey() | Retorna a última (maior) chave neste mapa ordenado. |
| V | pollFirstEntry() | Remove e retorna uma entrada mapeada para a primeira (menor) chave neste mapa ou nulo se o mapa estiver vazio. |
| V | pollLastEntry() | Remove e retorna uma entrada mapeada para a última (maior) chave neste mapa ou nulo se o mapa estiver vazio. |
| V | put(K key, V value) | Associa o valor especificado com a chave especificada neste mapa (operando atomicamente). |
| V | remove(Object key) | Remove o mapeamento para a chave especificada deste mapa, se presente. |
| int | size() | Retorna o número de mapeamentos |
| SortedMap<K, V> | subMap(K fromKey, K toKey) | Retorna uma visão de mapa deste mapa, cujas chaves variam de fromKey, inclusive, a toKey, exclusivo. |
| SortedMap<K, V> | subMap(K fromKey, K toKey) | Retorna uma visão de mapa deste mapa, cujas chaves variam de fromKey, inclusive, a toKey, exclusivo. |
| SortedMap<K, V> | tailMap(K fromKey) | Retorna uma visão de cauda de mapa do mapa deste mapa, cujas chaves são maiores ou iguais a fromKey. |
| SortedMap<K, V> | tailMap(K fromKey) | Retorna uma visão de cauda de mapa do mapa deste mapa, cujas chaves são maiores ou iguais a fromKey. |
| Collection<V> | values() | Retorna uma coleção de todos os valores deste mapa. |

## Métodos da interface Map.Entry

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Método | Descrição |
| static <K extends Comparable<? super K>,V> Comparator<Map.Entry<K,V>> | comparingByKey() | Ele retorna um comparador que compara os objetos em ordem natural na chave. |
| static <K,V> Comparator<Map.Entry<K,V>> | comparingByKey(Comparator<? super K> cmp) | Ele retorna um comparador que compara os objetos por chave usando o Comparator fornecido. |
| static <K,V extends Comparable<? super V>> Comparator<Map.Entry<K,V>> | comparingByValue() | Ele retorna um comparador que compara os objetos em ordem natural de valor. |
| static <K,V> Comparator<Map.Entry<K,V>> | comparingByValue(Comparator<? super V> cmp) | Ele retorna um comparador que compara os objetos por valor usando o Comparator fornecido. |
| boolean | equals(Object o) | É usado para comparar o objeto especificado com os outros objetos existentes. |
| K | getKey() | É usado para obter uma chave. |
| V | getValue() | É usado para obter valor. |
| int | hashCode() | É usado para obter hashCode. |
| V | setValue(V value) | É usado para substituir o valor correspondente a esta entrada pelo valor especificado. |

## Métodos da Classe HashMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | clear() | Remove todas as associações de chave-valor deste mapa. |
| Object[] | clone() | Retorna uma cópia superficial deste HashMap. |
| boolean | containsKey(Object key) | Retorna true se este mapa contém uma associação para a chave especificada. |
| boolean | containsValue(Object value) | Retorna true se este mapa mapeia uma ou mais chaves para o valor especificado. |
| Set<Map.Entry<K, V>> | entrySet() | Retorna um conjunto contendo todas as entradas deste mapa. |
| boolean | equals(Object o) | Compara o objeto especificado com este mapa para igualdade. |
| V | get(Object key) | Retorna o valor ao qual a chave especificada está mapeada ou nulo se este mapa não contém a chave. |
| int | hashCode() | Retorna o código hash para este mapa. |
| boolean | isEmpty() | Retorna true se este mapa não contém nenhuma associação de chave-valor. |
| Set<K> | keySet() | Retorna um conjunto contendo todas as chaves deste mapa. |
| V | put(K key, V value) | Associa o valor especificado com a chave especificada neste mapa. |
| void | putAll(Map<? extends K,? extends V> m) | Copia todas as associações de chave-valor do mapa especificado para este mapa. |
| V | remove(Object key) | Remove a associação para a chave especificada deste mapa se estiver presente. |
| int | size() | Retorna o número de associações de chave-valor neste mapa. |
| Collection<V> | values() | Retorna uma coleção contendo todos os valores deste mapa. |

## Métodos da Classe LinkedHashMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | clear() | Remove todos os mapeamentos deste LinkedHashMap. |
| boolean | containsKey(Object key) | Retorna true se esta LinkedHashMap contiver um mapeamento para a chave especificada. |
| boolean | containsValue(Object value) | Retorna true se esta LinkedHashMap contiver um ou mais mapeamentos para o valor especificado. |
| Set<Map.Entry<K, V>> | entrySet() | Retorna um conjunto de mapeamentos contidos nesta LinkedHashMap. |
| V | get(Object key) | Retorna o valor associado à chave especificada, ou nulo se não houver nenhuma correspondência. |
| V | getOrDefault(Object key, V defaultValue) | Retorna o valor ao qual a chave especificada está mapeada, ou o valor padrão especificado se esta LinkedHashMap não contiver a chave. |
| boolean | isEmpty() | Retorna true se esta LinkedHashMap não contiver nenhum mapeamento. |
| Set<K> | keySet() | Retorna um conjunto de chaves contidas neste LinkedHashMap. |
| V | put(K key, V value) | Associa o valor especificado à chave especificada nesta LinkedHashMap. |
| void | putAll(Map<? extends K, ? extends V> m) | Copia todos os mapeamentos do mapa especificado para esta LinkedHashMap. |
| V | remove(Object key) | Remove o mapeamento para a chave especificada desta LinkedHashMap, se estiver presente. |
| void | setOrder(boolean accessOrder) | Se accessOrder for true, a ordem será baseada na última vez em que o elemento foi acessado. Caso contrário, a ordem será baseada na ordem de inserção. |
| int | size() | Retorna o número de mapeamentos em esta LinkedHashMap. |
| Collection<V> | values() | Retorna uma coleção de valores contidos nesta LinkedHashMap. |

## Métodos da Classe WeakHashMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | clear() | Remove todos os mapeamentos da tabela hash. |
| boolean | containsKey(Object key) | Retorna verdadeiro se a tabela hash contiver a chave especificada. |
| boolean | containsValue(Object value) | Retorna verdadeiro se a tabela hash contiver um ou mais valores associados à chave especificada. |
| Set<Map.Entry<K, V>> | entrySet() | Retorna um conjunto de pares chave-valor contidos na tabela hash. |
| boolean | isEmpty() | Retorna verdadeiro se a tabela hash estiver vazia. |
| Set<K> | keySet() | Retorna um conjunto de chaves contidas na tabela hash. |
| V | put(K key, V value) | Associa o valor especificado com a chave especificada na tabela hash. |
| void | putAll(Map<? extends K,? extends V> m) | Copia todos os mapeamentos do mapa especificado para a tabela hash. |
| V | remove(Object key) | Remove o mapeamento para a chave especificada, se existir. |
| boolean | remove(Object key, Object value) | Remove o mapeamento para a chave especificada apenas se ele estiver associado ao valor especificado. |
| int | size() | Retorna o número de mapeamentos na tabela hash. |
| Collection<V> | values() | Retorna uma coleção de valores contidos na tabela hash. |

## Métodos da Classe IndentityHashMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | clear() | Remove todos os mapeamentos da tabela hash. |
| boolean | containsKey(Object key) | Retorna verdadeiro se a tabela hash contiver a chave especificada. |
| boolean | containsValue(Object value) | Retorna verdadeiro se a tabela hash contiver um ou mais valores associados à chave especificada. |
| Set<Map.Entry<K, V>> | entrySet() | Retorna um conjunto de pares chave-valor contidos na tabela hash. |
| boolean | isEmpty() | Retorna verdadeiro se a tabela hash estiver vazia. |
| Set<K> | keySet() | Retorna um conjunto de chaves contidas na tabela hash. |
| V | put(K key, V value) | Associa o valor especificado com a chave especificada na tabela hash. |
| void | putAll(Map<? extends K,? extends V> m) | Copia todos os mapeamentos do mapa especificado para a tabela hash. |
| V | remove(Object key) | Remove o mapeamento para a chave especificada, se existir. |
| int | size() | Retorna o número de mapeamentos na tabela hash. |
| Collection<V> | values() | Retorna uma coleção de valores contidos na tabela hash. |

## Métodos da Classe EnumMap

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | clear() | Remove todos os mapeamentos da enumeração. |
| Object | clone() | Retorna uma cópia rasa da instância EnumMap. |
| boolean | containsKey(Object key) | Retorna verdadeiro se esta enumeração contém a chave especificada. |
| boolean | containsValue(Object value) | Retorna verdadeiro se esta enumeração mapeia um ou mais valores para a chave especificada. |
| Set<Map.Entry<K, V>> | entrySet() | Retorna um conjunto de pares chave-valor contidos na enumeração. |
| boolean | equals(Object o) | Compara esta enumeração com o objeto especificado para igualdade. |
| V | get(Object key) | Retorna o valor ao qual esta enumeração mapeia a chave especificada. |
| int | hashCode() | Retorna o código hash para esta enumeração. |
| boolean | isEmpty() | Retorna verdadeiro se esta enumeração estiver vazia. |
| Set<K> | keySet() | Retorna um conjunto de chaves contidas na enumeração. |
| V | put(K key, V value) | Associa o valor especificado com a chave especificada nesta enumeração. |
| void | putAll(Map<? extends K,? extends V> m) | Copia todos os mapeamentos do mapa especificado para esta enumeração. |
| V | remove(Object key) | Remove o mapeamento para a chave especificada, se existir. |
| int | size() | Retorna o número de mapeamentos nesta enumeração. |
| Collection<V> | values() | Retorna uma coleção de valores contidos nesta enumeração. |

## Métodos da Classe Dictionary

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| Object | get(Object key) | Retorna o valor associado à chave especificada nesta tabela de hash. |
| Enumeration<Object> | elements() | Retorna uma enumeração dos valores nesta tabela de hash. |
| Enumeration<Object> | keys() | Retorna uma enumeração das chaves nesta tabela de hash. |
| int | size() | Retorna o número de entradas nesta tabela de hash. |
| boolean | isEmpty() | Retorna true se esta tabela de hash estiver vazia. |
| Object | put(Object key, Object value) | Associa o valor especificado à chave especificada nesta tabela de hash. |
| Object | remove(Object key) | Remove a entrada para a chave especificada desta tabela de hash, se presente. |
| void | clear() | Remove todas as entradas desta tabela de hash. |

## Métodos da Classe HashTable

|  |  |  |
| --- | --- | --- |
| Tipo de Dado Retornado | Métodos | Descrição |
| void | clear() | Remove todas as entradas desta tabela de hash. |
| Object | clone() | Retorna uma cópia superficial desta tabela de hash. |
| boolean | contains(Object value) | Retorna true se algum valor nesta tabela de hash for igual ao objeto especificado. |
| boolean | containsKey(Object key) | Retorna true se esta tabela de hash contiver a chave especificada. |
| boolean | containsValue(Object value) | Retorna true se algum valor nesta tabela de hash for igual ao objeto especificado. |
| Enumeration<V> | elements() | Retorna uma enumeração dos valores nesta tabela de hash. |
| V | get(Object key) | Retorna o valor associado à chave especificada nesta tabela de hash. |
| boolean | isEmpty() | Retorna true se esta tabela de hash estiver vazia. |
| Enumeration<K> | keys() | Retorna uma enumeração das chaves nesta tabela de hash. |
| V | put(K key, V value) | Associa o valor especificado à chave especificada nesta tabela de hash. |
| void | putAll(Map<? extends K, ? extends V> t) | Copia todas as associações do mapa especificado para esta tabela de hash. |
| V | remove(Object key) | Remove a entrada para a chave especificada desta tabela de hash, se presente. |
| int | size() | Retorna o número de entradas nesta tabela de hash. |
| String | toString() | Retorna uma string que representa esta tabela de hash. |