Índice

[Optional 1](#_butpk9qu101k)

[Comandos 1](#_8q0jtr46kxla)

[Stream 2](#_5o2tndg76ro)

[Comandos 3](#_togp0w4ad917)

[Tabela HEX Caracteres 4](#_hqvwvep88e6i)

[Expressão lambda 4](#_4rfzxs6go3st)

[Links e afins 5](#_v49ndfn8hwjl)

[Dúvidas 5](#_huow9eiz5of6)

# 

**Assuntos: JAVA Optional, JAVA Stream.**

# Optional

Pega um recurso, encapsula o retorno dos métodos deste recurso, e retorna uma espécie de boolean caso um valor esteja presente ou não.

Analogia: caixa de papelão, onde se guarda um objeto para se evitar erros neste objeto.

O propósito principal do Optional é lidar com situações onde um valor pode ser nulo, indicando que talvez não exista um valor presente para determinada variável ou objeto.

## Comandos

**.map( )** Comando que mapeia o optional, e retorna uma resposta caso o valor esteja presente ou não.

**.of( x )** É um método do optional, ele retorna um optional com o valor x fornecido. Não pode ser nulo, a não ser que use o ofNullable().

**.filter( )** Filtra o optional de acordo com uma condição.

**.startsWith()** Define que o objeto escolhido inicie com um caractere.

exemplo: Se colocar .startsWith("T") irá verificar se a String começa com T.

**.orElseThrown( )** Como uma espécie de “else”, onde ele executa um comando caso as condições anteriores não sejam correspondidas.

Entenda o **.startsWith()** e **.orElseThrown( )** como um if else, caso o .startsWith() não seja atendido ele executa o .orElseThrow. Eles sempre aparecem após usar o **.map**.

**.empty( )** Retorna uma instância do optional vazia.

**.ofNullable( )** Se um valor estiver presente, retorne um optional com o valor. Caso contrário, retorna um optional vazio.

**.isPresent( )**  Checa se um valor está presente. Retorna true ou false.

**.get( )** Se houver um valor presente, retorna o valor, caso contrário, lança a exceção **NuSuchElementException**.

**.ifPresent( )** Se um valor estiver presente, executa uma ação. Caso contrário, não executa nada.

# 

# Stream

Uma API, que permite o processamento de dados de forma declarativa.

O stream mapeia uma collection, faz algumas operações, e busca uma série de dados, para depois fazer uma execução. Semelhante a um optional, mas para collections.

Imagine que você tem uma esteira transportadora onde várias caixas passam por você. Cada caixa pode conter algo diferente dentro dela. Você está interessado em pegar apenas as caixas de uma cor específica ou talvez pegar as maiores ou menores caixas. A esteira transportadora é como uma Stream.

Em programação, uma Stream é como essa esteira transportadora. Ela permite que você processe uma sequência de elementos (como uma coleção de dados, por exemplo, uma lista) e realize operações neles, como filtrar, mapear, reduzir ou transformar esses elementos.

**Operações Intermediárias:**

São operações que podem ser executadas durante processamento da Stream, não possuindo um limitador de quantas operações serão executadas, e sempre dependendo da necessidade do tratamento de dados.

Ela sempre retorna seus dados tratados de acordo com sua função, prontos para serem acessados por uma nova operação intermediária, ou exibidos por uma operação terminal.

**Exemplos:** Map, Filter, Sorted, etc.

**Operações terminais:**

São operações incapazes de retornar dados para uma interface Stream, mas sim um resultado ou void.

Por sempre retornar um resultado final, não se pode ter duas ou mais operações terminais em uma mesma stream.

A operação terminal deve sempre ser o último argumento de uma stream.

**Exemplos:** forEach, collect, etc.

## Comandos

**.stream( )** Converte uma lista para um stream.

**.map( )** Passa por todos os elementos de uma collection, e capaz de causar transformações em cada um dos itens.

**.filter ( )** Comando para se delimitar filtros para uma lista.

**.sorted ( )** Comando para se sortear, ou seja, organizar, uma lista em ordem.

**.count( )** Conta a quantidade de elementos em uma Stream. Quantidade de índices.

**.collect ( )** Cria uma lista com todos os elementos processados na stream.

***x*.toString ( )** Verifica todas as Strings. ‘x’ pode ser alterado para um nome desejado.

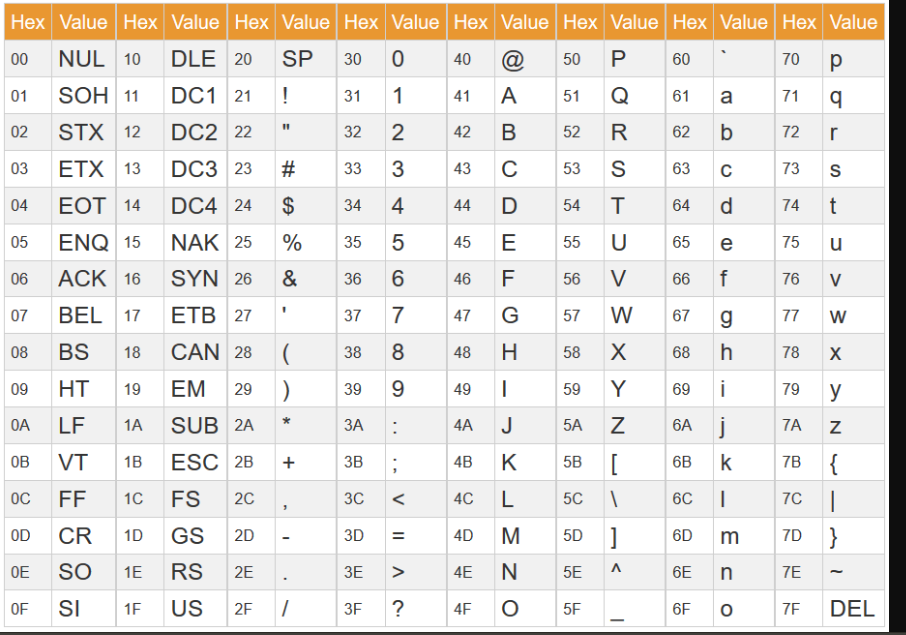
**.endsWith( )** Verifica o caractere final dos elementos. Ou seja, se colocarmos .endsWith(b), ele vai filtrar todos os elementos que tem como caractere final a letra b.

**.allMatch( )** Verifica se todos contem o que foi especificado, caso não tenha ele retorna false.

**.contains( )** Verifica se existe um elemento em uma stream.

**.anyMatch( )** Uma espécie de boolean, retorna true ou false para caso exista um elemento com determinadas características na stream.

### Tabela HEX Caracteres



# Expressão lambda

Expressão lambda é como uma “função” anônima. Não sendo, exatamente, uma função.

**Expressão lambda** é como uma receita compacta para fazer algo. É uma maneira mais concisa de escrever pequenas funções.

Pense nisso como uma receita em que você diz o que quer fazer, sem precisar escrever todo o passo a passo.

Por exemplo, se você quiser dobrar um número, a expressão lambda seria como uma receita que diz "pegue um número e o dobre". Isso torna o código mais curto e fácil de entender.

**( ) ->** Expressão lambda sem argumentos.

**generation ->** Expressão lambda cujo argumento se chama generation.

A Expressão lambda é definida por essa setinha “ -> “.

**Exemplo:**

**(resposta -> resposta.getNome( ))**

# Links e afins

# 

# Dúvidas

1. O orElseThrow sempre precisa retornar um erro ou pode retornar um sysout também?

O método orElseThrow em Java é projetado para lançar uma exceção quando o Optional está vazio. Não é apropriado retornar um System.out.println() diretamente.

Se deseja imprimir uma mensagem antes de lançar uma exceção, você pode fazer algo como:

optionalValue.orElseThrow(() -> {

System.out.println("Valor não encontrado!");

return new NoSuchElementException("Mensagem de exceção");

});

ou

Dá pra não lançar exception e só deixar o sysout tbm, nesse caso ao invés de orElseThrow use só: orElse

1. O que é um optional?

O Optional é uma espécie de contêiner ou wrapper que lida com a possibilidade de um valor estar ausente (nulo). Ele fornece métodos para manipular e acessar esse valor de maneira mais segura, evitando diretamente os problemas associados a valores nulos.

Ele encapsula um valor que pode existir ou não, fornecendo métodos para manipular e acessar esse valor de maneira segura.

1. O que é Stream?

Stream é como uma esteira que você pode usar para manipular uma lista de coisas (como números). Você pode fazer diferentes operações nessa esteira, como pegar só os números pares, dobrar todos eles, ou somá-los. É uma forma rápida e eficiente de lidar com muitos dados de uma vez.

É como uma forma de processar e transformar dados sem modificar a coleção original. Então, usando a metáfora da esteira, a esteira não altera os produtos que estão sendo transportados, mas permite que você faça diferentes operações nesses produtos conforme eles passam pela esteira, criando novos resultados.