# Peek – Dando uma olhada para saber como anda o código.

.filter(e -> !e.avaliacao().equalsIgnoreCase("N/A")).peek(e -> System.*out*.println("Primeiro filtro(N/A) " + e))  
  
.sorted(Comparator.*comparing*(DadosEpisodios::avaliacao).reversed())  
.peek(e -> System.*out*.println("Ordenação " + e))  
  
.limit(10)  
.peek(e -> System.*out*.println("Limite " + e))  
  
.map(e -> e.titulo().toUpperCase())  
.peek(e -> System.*out*.println("Mapeamento " + e))  
  
.forEach(System.*out*::println);

Aqui vamos usar o PEEK para saber oque vem sendo feito.

No nosso primeiro filtro ele vai pegar todo mundo que não é N/A

Assim podemos ver passo a passo do que minha coleção encadeada de dados está fazendo.

SAIDA:

Primeiro filtro(N/A)DadosEpisodios[titulo=The Female of the Species, numero=4, avaliacao=8.7, dataLancamento=2019-07-26]

# E a função Peek, o que é?

A função”peek” é uma operação intermediária de uma stream. Uma operação intermediária é aquela que processa os dados da stream, mas não a finaliza, ou seja, ela permite que mais operações sejam feitas depois dela. A função peek é usada para "espiar" os elementos da stream sem alterá-los, o que pode ser muito útil para depuração. Essa função vai receber cada elemento da stream e realizar alguma operação de efeito colateral (como imprimir o elemento), mas vai devolver o mesmo elemento sem alterar nada.

# Por que eu deveria usar isso?

O uso de Streams e a função peek podem te ajudar a escrever códigos mais elegantes, fáceis de ler, e diminuir a possibilidade de erros. Além disso, eles podem fazer seu código rodar mais rápido em certos casos, pois os Streams permitem a paralelização das operações, ou seja, executar várias operações ao mesmo tempo.

# Exemplos de código

Vamos ver agora um exemplo simples de como podemos usar a função peek com streams. Imagine que temos uma lista com números e queremos multiplicar cada um deles por 2 e depois somar os resultados.

# Exemplos de código

Vamos ver agora um exemplo simples de como podemos usar a função peek com streams. Imagine que temos uma lista com números e queremos multiplicar cada um deles por 2 e depois somar os resultados.

List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);

**int** soma = numeros.stream()

.peek(n -> System.out.println("Elemento: " + n))

.map(n -> n \* 2)

.peek(n -> System.out.println("Conteúdo depois do map: " + n))

.reduce(0, (total, numero) -> total + numero);

System.out.println("A soma dos números é: " + soma);

SAIDA: A soma dos números é 30

Isso nos ajuda a ter certeza de que o processo está sendo feito corretamente. Aqui conseguimos perceber que a soma dos números 2 + 4 + 6 + 8 + 10 foi calculada corretamente, resultando em 30. Ou seja, a função lambda em nosso método reduce está fazendo exatamente o que era planejado. E pudemos conferir isso, com o peek.

Esse é apenas um exemplo básico, mas lembre-se que a função peek pode ser usada em qualquer lugar dentro da sua stream, sempre que você quiser ver quais elementos estão passando por aquele ponto.