# Streams

são fluxos de dados.

// Fazendo uma lista  
 List<String> nomes = Arrays.*asList*("Fernanda", "Valeria", "Alleph", "Livia", "Larissa");  
  
 nomes.stream()  
 .sorted() // 1 Colocando em ordem  
 .limit(3) // Limitando para pegar apenas 3 nomes

.filter(n -> n.startsWith("A"))// Aqui estamos criando uma função lambda passando que o n é o parâmetro -> n = nomes.startWith começa com a letra A

.map(n -> n.toUpperCase())Mandando pegar o nome e transformar em maiusculo

.forEach(System.*out*::println); // 2 Imprimir cada um dos nomes.  
}

List estou iniciando uma lista de Strings

Passando tudo para uma variável chamada **nomes**

Nomes está recebendo um Array do tipo **asList**

* (Uma lista estática que não pode aumentar ou diminuir o tamanho, mas podemos modificar oque está dentro)

Agora estamos encadeando as informações e gerando um fluxo que vai.

.sorted = colocar os nomes em ordem.

.limit(3) pegando no máximo 3 opções

.filter(n -> n.startsWith("A")) Pegando somente os nomes coma letra A

.map(n -> n.ToUpperCase() Mandando transformar oque ele pegou em maiúsculo.

forEach(Imprimir os nomes uma a um)

Ou seja, seguindo a leitura.

Primeiro coloca tudo em ordem

Depois pega os que estão em ordem e me da no máximo 3 nomes

Desses 3 nomes me da somente os que começam com a letra A

Agora pega esse resultado e coloca em Maisculo depois imprime um de cada vez.

Lembrar que um streams, gera duas operações

* Intermediaria
  + As operações intermediárias são aquelas que podem ser aplicadas em uma stream e retornam uma nova stream como resultado. Essas operações não são executadas imediatamente, mas apenas quando uma operação final é chamada.
* Operações finais.
  + As operações finais são aquelas que encerram a stream e retornam um resultado concreto. Algumas operações finais comuns são o forEach, collect e count.
  + Vamos ver alguns exemplos de operações finais: ForEach: permite executar uma ação em cada elemento da stream. Por exemplo, podemos imprimir cada elemento da lista.

List<**String**> nomes = Arrays.**asList**("João", "Maria", "Pedro", "Ana");

nomes.**stream**()

.**forEach**(nome -> System.out.**println**("Olá, " + nome + "!"));

// Output:

// Olá, João!

// Olá, Maria!

// Olá, Pedro!

// Olá, Ana!

# Outros exemplos de Streams

Por fim, As streams e suas operações intermediárias e finais são uma forma poderosa de manipular coleções de dados no Java de maneira mais eficiente e concisa. Elas permitem escrever código mais legível e expressivo, facilitando o processamento e a transformação de dados.

Além das operações intermediárias e finais mencionadas, existem muitas outras disponíveis, como distinct (que remove elementos duplicados), limit (que limita o número de elementos da stream), skip (que pula os primeiros elementos da stream), reduce (que combina os elementos da stream em um único resultado) e muitas outras.

Streams no Java são uma poderosa ferramenta para manipulação de coleções de dados. As operações intermediárias permitem filtrar, mapear e ordenar os elementos da stream, enquanto as operações finais encerram a stream e retornam um resultado concreto. Combinando essas operações, podemos realizar transformações e ações de forma concisa e eficiente.

# Segue outro exemplo de lista

public class Main {

public static void main(String[] args) {

// Isso não funciona como esperado, pois o array é tratado como um único elemento

int[] numerosArray = {1, 2, 3, 4, 5};

List<int[]> listaNumeros = Arrays.asList(numerosArray);

System.out.println("Tamanho da lista: " + listaNumeros.size()); // Output: 1

// Solução: Usar a classe wrapper Integer

Integer[] numerosWrapperArray = {1, 2, 3, 4, 5};

List<Integer> listaNumerosWrapper = Arrays.asList(numerosWrapperArray);

System.out.println("Tamanho da lista: " + listaNumerosWrapper.size()); // Output: 5

}

}