O que é uma Stream?

Uma Stream é uma sequência de elementos que pode ser processada em paralelo ou sequencialmente. Streams permitem realizar operações como filtro, mapeamento, redução, coleta e iteração de maneira declarativa.

Criação de Streams

Você pode criar streams de várias fontes, como coleções, arrays ou até mesmo arquivos. Por exemplo:

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class Example {
    public static void main(String[] args) {
        // De uma lista
        List<String> list = Arrays.asList("a", "b", "c");
        Stream<String> stream1 = list.stream();

        // De um array
        String[] array = {"x", "y", "z"};
        Stream<String> stream2 = Arrays.stream(array);

        // De valores específicos
        Stream<String> stream3 = Stream.of("m", "n", "o");
    }
}
```

Operações Intermediárias e Terminais

As operações em streams são de dois tipos: intermediárias e terminais.

- **Operações Intermediárias**: Produzem outra stream e são “lazy”, ou seja, não executam até que uma operação terminal seja invocada. Exemplos: `filter()`, `map()`, `sorted()`.

- **Operações Terminais**: Produzem um resultado ou efeito colateral e encerram o processamento da stream. Exemplos: `forEach()`, `collect()`, `reduce()`.

Exemplos de Uso

Filtrando Elementos

```
List<String> words = Arrays.asList("apple", "banana", "cherry", "date");
List<String> filtered = words.stream()
                             .filter(word -> word.startsWith("b"))
                             .collect(Collectors.toList());

System.out.println(filtered); // Output: [banana]
```

Mapeando Elementos

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4);
List<Integer> squared = numbers.stream()
                               .map(n -> n * n)
                               .collect(Collectors.toList());

System.out.println(squared); // Output: [1, 4, 9, 16]
```

Reduzindo Elementos

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4);
int sum = numbers.stream()
                 .reduce(0, (a, b) -> a + b);

System.out.println(sum); // Output: 10
```

Benefícios das Streams

- **Legibilidade**: Código mais claro e conciso.
- **Paralelismo**: Fácil de paralelizar operações em grandes coleções de dados.
- **Imutabilidade**: Streams são imutáveis, o que ajuda na programação funcional.

A API de Streams é uma ferramenta poderosa para trabalhar com coleções e outros conjuntos de dados no Java, tornando as operações complexas mais intuitivas e legíveis.

Se tiver mais dúvidas ou precisar de mais exemplos, estou aqui para ajudar!