Stream Collection Java

📘 Aula — Streams da API Collections (Java)

⏱️ **Duração:** 1 hora
🎯 **Objetivo:** Ensinar Streams **de forma prática**, mostrando **como usar e executar** no dia a dia profissional.

🧠 PARTE 1 — TEORIA (≈ 20 minutos)

🔹 O que são Streams?

Streams são uma **API funcional do Java** (desde o Java 8) usada para **processar coleções** de forma:

- Declarativa
- Imutável
- Encadeável
- Expressiva

📌 *Stream não armazena dados*, apenas **processa dados** de uma Collection.

🔹 Vantagens dos Streams

✅ Código mais curto e limpo
✅ Menos `for`, `if` e variáveis auxiliares
✅ Fácil leitura
✅ Facilita manutenção
✅ Excelente para regras de negócio
✅ Muito usado em projetos reais (APIs, sistemas, relatórios)

🔹 Estrutura básica

```
java
colecao.stream()
    .operacaoIntermediaria()
    .operacaoFinal();
```

🔹 Operações Intermediárias

- `filter`
- `map`
- `sorted`
- `distinct`

🔹 Operações Finais

- `forEach`
- `collect`
- `sum`
- `count`
- `anyMatch`

- `findFirst`

🧪 PARTE 2 — EXEMPLOS PRÁTICOS (≈ 40 minutos)

✅ EXEMPLO 1 — Filtrar usuários ativos

📌 Classe Usuario

```
java
public class Usuario {
    private String nome;
    private boolean ativo;

    public Usuario(String nome, boolean ativo) {
        this.nome = nome;
        this.ativo = ativo;
    }

    public String getNome() {
        return nome;
    }

    public boolean isAtivo() {
        return ativo;
    }
}
```

▶️ Classe Main

```java
import java.util.List;

public class Main {
    public static void main(String[] args) {

        List<Usuario> usuarios = List.of(
                new Usuario("Alex", true),
                new Usuario("Maria", false),
                new Usuario("João", true)
        );

        List<Usuario> ativos = usuarios.stream()
                .filter(Usuario::isAtivo)
                .toList();

        ativos.forEach(u -> System.out.println(u.getNome()));
    }
}
```

🧠 **Explicação:**
Filtra apenas usuários ativos sem `for` ou `if`.

✅ EXEMPLO 2 — Converter produtos em lista de nomes

📌 Classe Produto

```
java
public class Produto {
    private String nome;
    private double preco;

    public Produto(String nome, double preco) {
        this.nome = nome;
        this.preco = preco;
    }

    public String getNome() {
        return nome;
    }
}
```

▶️ Classe Main

```
java
import java.util.List;

public class Main {
    public static void main(String[] args) {

        List<Produto> produtos = List.of(
                new Produto("Notebook", 4500),
                new Produto("Mouse", 120),
                new Produto("Teclado", 300)
        );

        List<String> nomes = produtos.stream()
                .map(Produto::getNome)
                .toList();

        nomes.forEach(System.out::println);
    }
}
```

🧠 **Explicação:**
`map` transforma objetos em outro tipo (muito usado em DTO).

✅ EXEMPLO 3 — Somar valor total de pedidos

📌 Classe Pedido

```
java
public class Pedido {
    private double valor;

    public Pedido(double valor) {
        this.valor = valor;
    }

    public double getValor() {
        return valor;
    }
}
```

▶️ Classe Main

```
java
import java.util.List;

public class Main {
    public static void main(String[] args) {

        List<Pedido> pedidos = List.of(
                new Pedido(150),
                new Pedido(300),
                new Pedido(550)
        );

        double total = pedidos.stream()
                .mapToDouble(Pedido::getValor)
                .sum();

        System.out.println("Total vendido: R$ " + total);
    }
}
```

🧠 **Explicação:**
Ideal para **relatórios financeiros e dashboards**.

✅ EXEMPLO 4 — Agrupar funcionários por setor

📌 Classe Funcionario

```
java
public class Funcionario {
    private String nome;
    private String setor;

    public Funcionario(String nome, String setor) {
        this.nome = nome;
        this.setor = setor;
    }

    public String getSetor() {
        return setor;
    }

    public String getNome() {
        return nome;
    }
}
```

▶️ Classe Main

```
java
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class Main {
    public static void main(String[] args) {

        List<Funcionario> funcionarios = List.of(
                new Funcionario("Ana", "TI"),
                new Funcionario("Carlos", "RH"),
                new Funcionario("Beatriz", "TI")
        );

        Map<String, List<Funcionario>> agrupado =
                funcionarios.stream()

.collect(Collectors.groupingBy(Funcionario::getSetor));

        agrupado.forEach((setor, lista) -> {
            System.out.println("Setor: " + setor);
            lista.forEach(f -> System.out.println(" - " +
f.getNome()));
        });
    }
}
```

🧠 Explicação:

Muito usado em **relatórios, BI e sistemas administrativos**.

✅ EXEMPLO 5 — Verificar pedido acima de R$ 10.000

▶️ Classe Main

```
java
import java.util.List;

public class Main {
    public static void main(String[] args) {

        List<Pedido> pedidos = List.of(
                new Pedido(500),
                new Pedido(12000),
                new Pedido(800)
        );

        boolean existePedidoAlto = pedidos.stream()
                .anyMatch(p -> p.getValor() > 10000);

        System.out.println("Existe pedido acima de 10 mil? " +
existePedidoAlto);
    }
}
```

🧠 **Explicação:**
Usado para **validações de regra de negócio**.

🧠 Fechamento da Aula

- ✔️ Streams são padrão no Java moderno
- ✔️ Facilitam leitura e manutenção
- ✔️ Substituem laços complexos
- ✔️ Muito cobrados em entrevistas

📚 Exercícios sugeridos

1. Filtrar clientes maiores de idade
2. Calcular média de salários
3. Agrupar pedidos por status
4. Converter lista de entidades em DTO