Conceitos de Programação Funcional

Declarar invés de instruir:

O cerne da programação funcional é desenvolver o software usando funções que sejam declarativas e não imperativos. Da pra entender isso pensando que você pode ***mandar seu código fazer algo*** invés de ***passar cada mínima instrução pra ele***:

Comandos linha a linha para calcular um total.

```
public double calcularTotal(Cardapio cardapio){
    double total= 0;

    for (ItemPedido item: itens) {
        var shake = item.getShake();
        var qtdShake = item.getQuantidade();
        var adicionais = shake.getAdicionais();

        var precoBase = cardapio.getPrecos().get(shake.getBase());
        var precoComQuantidade = precoBase * (precoBase * shake.getTipoTamanho().multiplicador);
        var totalAdicionais = adicionais.stream().map(adicional -> cardapio.getPrecos().get(adicional))
                .reduce(Double::sum).orElse( other: 0.0);

        total += (precoComQuantidade + totalAdicionais) * qtdShake;
    }

    return total;
}
```

Cada instrução complexa é definida com uma declaração. ***Filtra, acha, pega ou lança uma exceção***

```
ItemPedido pedidoAtualizado = itens.stream()
        .filter(itemPedido -> itemPedido.equals(itemPedidoRemovido))
        .findAny()
        .orElseThrow();
```

Funções puras e imutabilidade:

São funções que ***sempre retornam o mesmo valor de acordo com os parâmetros de entrada***, ela não pode produzir resultados diferentes para parâmetros iguais. Ela também ***não deve alterar uma variável que está fora do escopo dela***, isso seria causar efeito colateral:

Não importa quantas vezes eu execute o reduce. Eu sempre vou ter o mesmo resultado para esse Set específico

```
Set<Integer> setInteger = new HashSet<>(Arrays.asList(1,2,3,4));

Integer total = setInteger.stream().reduce(Integer::sum).orElse( other: 0);
```

```
10
10
10
10
```

O **elemento fonte da função permanece inalterado**. Enquanto **um novo elemento é fornecido** com as mudanças feitas

```
List<Integer> listaInteger = new ArrayList<>(Arrays.asList(8,5,6,10));
Lista original: [8, 5, 6, 10]

List<Integer> listaOrdenada = listaInteger.stream().sorted().toList();
Nova lista ordenada: [5, 6, 8, 10]
Lista original permanece o mesmo: [8, 5, 6, 10]
```

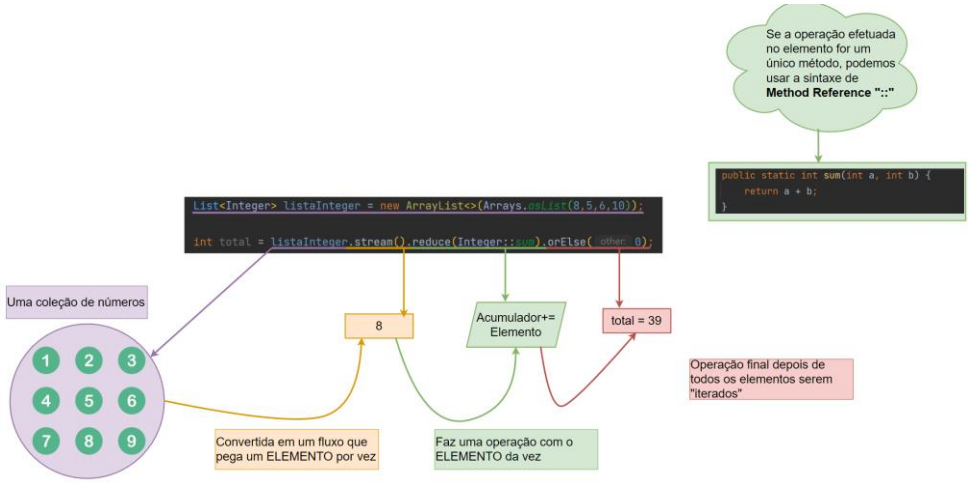
A imutabilidade está relacionada ao fato de não oferecer danos colaterais, então é uma boa prática retornar um novo valor invés de alterar o valor fonte.

Composição de funções:

Funções podem ser **compostas afim e entregar um resultado mais completo**, não é diferente do que já vimos antes, é **simplesmente o fato de uma função poder ser encadeada a outra**. Os exemplos acima já tratam isso.

Streams – Java Funcional

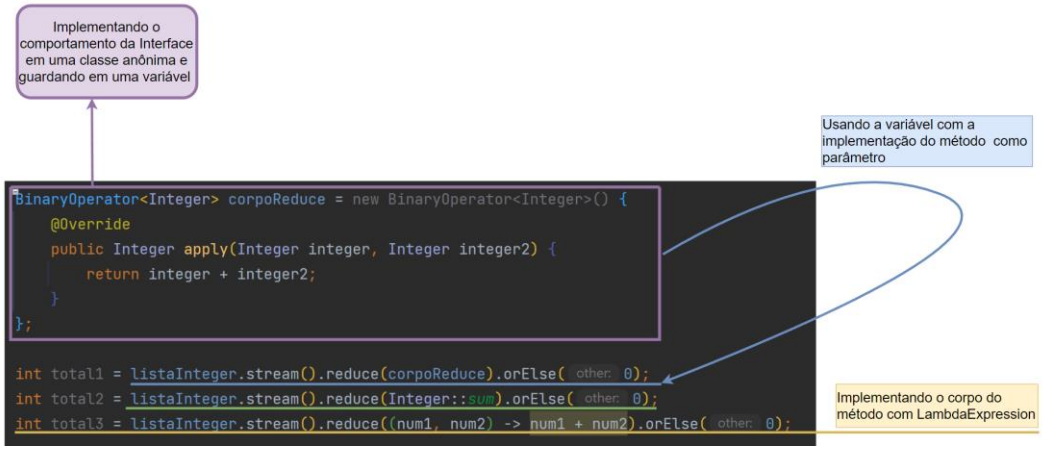
A implementação de Java que garantiu um bom uso do paradigma funcional foi a **Stream**. Stream é uma operação que **converte uma coleção/mapa/dados** em um **FLUXO** que vai **tratar UM a UM** desses **elementos**:



Lambda Expression – Java Funcional

Uma **expressão lambda nada mais é do que um jeito simples de definir o corpo de um método.** Métodos funcionais **usam Interfaces funcionais**, essas Interfaces **geralmente possuem uma única assinatura de método**.

Então quando usamos a sintaxe de lambda, estamos **implementando o comportamento daquele único método em uma classe anônima**, o compilador entende então que esse comportamento é o corpo do método esperado pela Interface funcional:



Method Reference – Java Funcional

A **referência de método** é uma sintaxe ***especial de lambda expression*** que faz referência a um método de uma classe durante uma operação funcional. Ela enxuga o código e traz mais expressividade para ele:

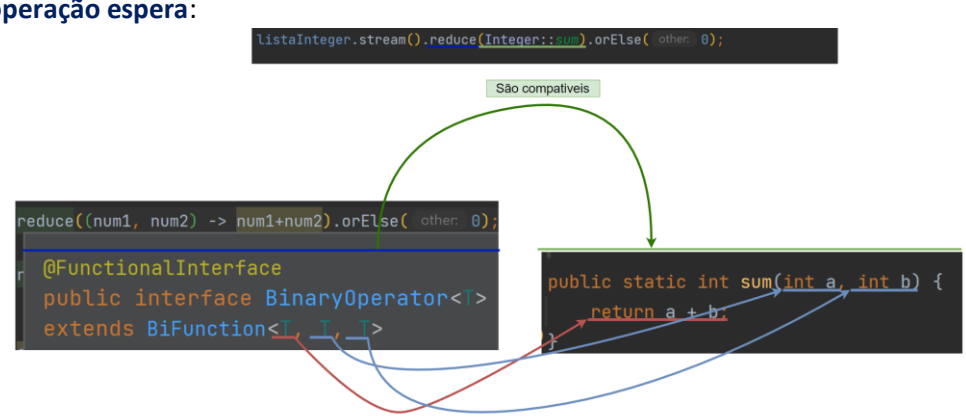
```
listaInteger.stream().reduce((num1, num2) -> num1+num2).orElse( other: 0);

listaInteger.stream().reduce(Integer::sum).orElse( other: 0);
```

```
pessoas.stream().forEach(pessoa -> System.out.println(pessoa));

pessoas.stream().forEach(System.out::println);
```

O **method reference** precisa atender a certas condições para ser usado, isso significa que, a **referência de método precisa receber os mesmos parâmetros e devolver o mesmo valor que a operação espera**:



O Compilador sempre vai te ajudar a substituir um lambda por um ***method reference*** quando der.