Introdução

A versão 8 do Java trouxe muitas novidades e melhorias focadas ao paradigma funcional de programação.

Programação funcional é um paradigma de programação completamente diferente da **Orientação a Objetos**, mas que o Java, como linguagem orientada a objetos, pode tirar vantagem.

Métodos default

A partir da versão 8 da linguagem podemos implementar métodos em interfaces para definir um comportamento padrão para as classes que as implementam:

```java
public interface Conta {
    default String getIban() {
        return "";
    }

    double calcularSaldo();
}

import java.util.List;

public class ContaPoupanca  implements Conta {
    private double saldo;
    private double taxaDeJuros;

    @Override
    public double calcularSaldo() {
        return saldo * (taxaDeJuros + 1);
    }
}
```

Dessa forma um objeto da classe "**ContaPoupanca**" terá o método "**getIban**" disponível, mesmo com sua implementação padrão

Expressões lambda

Expressões lambdas são essencialmente funções que são tratadas como um valor.
Funções são parecidas com métodos, exceto que elas não pertencem a nenhuma classe e podem ser consideradas como um valor, sendo possível serem passadas como argumentos para métodos, como se fossem uma variável ou objeto.

Em um cenário em que precisamos percorrer uma lista de objetos é comum usar o laço "for":

```java
List<Animal> animais = List. of(new Cachorro(), new
Passaro(), new Tubarao());

for (Animal animal : animais) {
    System. out.println(animal.getEspecie());
}
```

As listas em Java também implementam o método "forEach" em que podemos passar uma expressão lambda como argumento:

```java
animais.forEach((Animal animal) -> System.out.println(animal.getEspecie()));
```

No caso anterior, o método "**forEach**" recebe uma expressão lambda, ou seja, uma função que recebe um animal como argumento e imprime na tela a espécie do animal. O resultado do exemplo acima será o mesmo do laço "**for**", só que o código é mais conciso.

Como as expressões lambdas podem ser consideradas como valores, podemos atribuí-las a uma variável. No caso do método "**forEach**", o tipo do argumento deve ser "**Consumer**":

```
List<Animal> animais = List. of(new Cachorro(), new Passaro(), new
Tubarao());
Consumer<Animal> mostrador = (Animal animal) -> {
   System. out.println(animal.getEspecie());
};
animais.forEach(mostrador);
// Cachorro
// Pássaro
// Tubarão
```


Streams

Streams do Java é um recurso usado **para facilitar a manipulação de coleções de dados**, onde métodos de busca, mapeamento, ordenação, filtragem, contagem, entre outros estão implementados.

O seguinte **artigo** da **Oracle** descreve em detalhes as funcionalidades da biblioteca de Streams:

https://www.oracle.com/br/technical-resources/articles/java-stream-api.html

Nesse link está disponível a documentação online da biblioteca:

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

Filtragem

O método filter da biblioteca de Streams permite fazer a filtragem de elementos com base em um predicado. Predicado é uma expressão lambda que retorna um valor booleano:

```
List<Animal> animais =  new ArrayList<>();

animais.add(new Cachorro());
animais.add(new Passaro());
animais.add(new Tubarao());
animais.add(new Cachorro());
animais.add(new Passaro());
animais.add(new Tubarao());

List<Animal> cachorros = animais
       .stream()
       .filter((Animal animal) ->
animal.getEspecie().equals( "Cachorro")).toList();

cachorros.forEach(System. out::println);
// Animal{especie='Cachorro', formaDeLocomocao='Cachorros andam'}
// Animal{especie='Cachorro', formaDeLocomocao='Cachorros andam'}
```

No caso acima estamos criando uma lista de "**Animal**" e depois filtrando as instâncias de "**Cachorro**":

Mapeamento

Mapeamento se refere à capacidade de transformarmos objetos em outros. Vamos à um exemplo:

```
public class Pessoa {
   private String nome;
   private String sobrenome;

   public Pessoa(String nome, String sobrenome) {
        this.nome = nome;
        this.sobrenome = sobrenome;
   }

   // getters e setters
}

Pessoa p1 =  new Pessoa("Lucas", "Santos");
Pessoa p2 =  new Pessoa("Rafael", "Luiz");
Pessoa p3 =  new Pessoa("Claudia", "Rodrigues");

List<Pessoa> pessoas = List. of(p1, p2, p3);

List<String> nomesCompletos = pessoas
       .stream()
       .map((Pessoa pessoa) -> pessoa.getNome() +   " " +
pessoa.getSobrenome()).toList();

System.out.println(nomesCompletos);
```

No caso anterior, temos uma classe "**Pessoa**" que possui os atributos "**nome**" e "**sobrenome**". Então criamos uma lista com 3 objetos de "**Pessoa**". Usando a biblioteca de Streams do Java, foi chamado o método "map" que cria uma lista de String a partir da concatenação de "**nome**" e "**sobrenome**".

Podemos dizer que transformamos uma lista de "**Pessoa**" para uma lista de String.

Redução

A biblioteca de Streams do Java permite reduzir uma lista para um valor, de acordo com uma função transformadora:

```
public class Produto {
   private String descricao;
   private double preco;

   // Getters e Setters
}
```



```
Produto celular = new Produto("Celular", 1499.99);
Produto tv = new Produto("TV", 2099.99);
Produto videogame = new Produto("Videogame", 3299.99);

List<Produto> produtos = List.of(celular, tv, videogame);

double valorTotal =
produtos.stream().mapToDouble(Produto::getPreco).sum();

System.out.println(valorTotal);
```

No cenário acima, criamos uma lista de "**Produtos**", informando a descrição e o preço de cada um. Para obter a soma dos preços basta chamar o método "**mapToDouble**", informar o método que queremos usar como função transformadora e obter a soma por meio da chamada do método "**sum**".

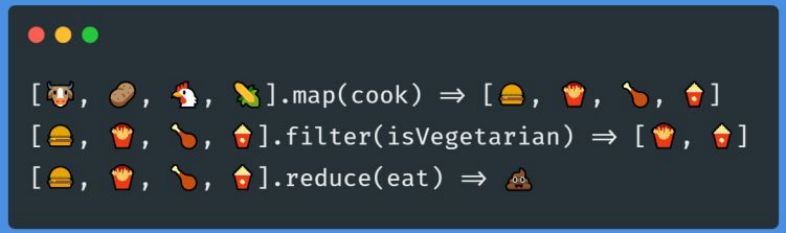
O que acabamos de fazer foi uma "**Redução**" (ou ***Reduce***), ou seja, transformamos um conjunto de dados em um único valor.

Esse link mostra outras formas de trabalharmos com somas de valores em uma lista:

https://www.baeldung.com/java-stream-sum

Filter, Map, Reduce

As 3 operações mencionadas são conhecidas por "**Filter, Map, Reduce**" e são as operações mais comuns quando lidamos com lista de dados. Em um cenário do mundo real as usamos quando necessitamos obter relatórios de vendas, produtos, arquivos, etc...



https://codeburst.io/map-filter-and-reduce-in-javascript-728f2b9ace8

A imagem acima exemplifica bem visualmente as três operações.