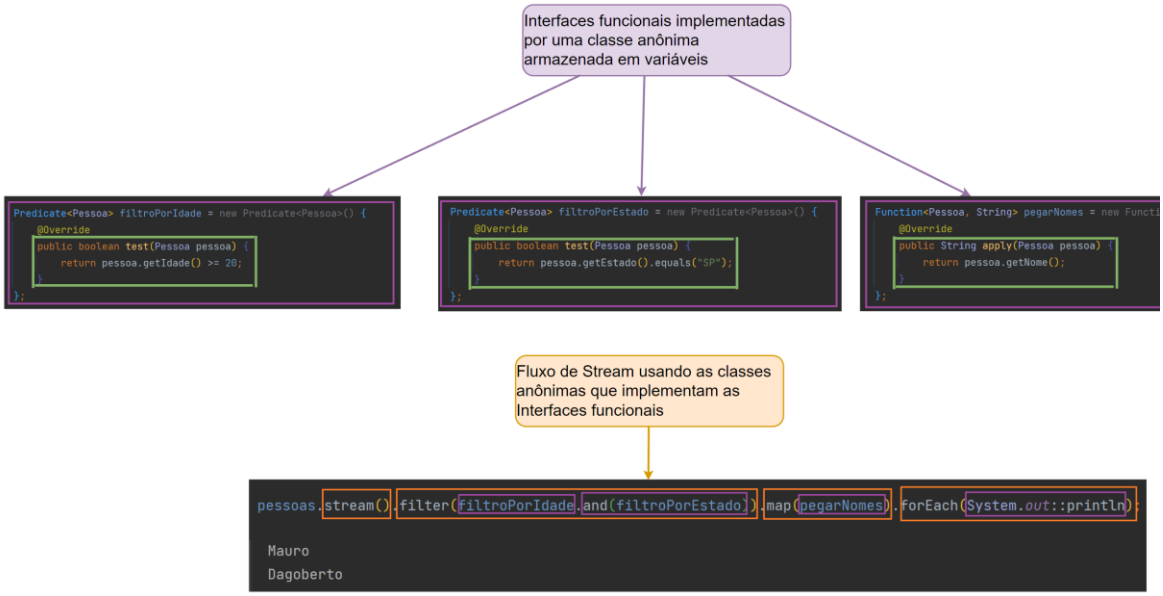
Interfaces Funcionais – Java Funcional

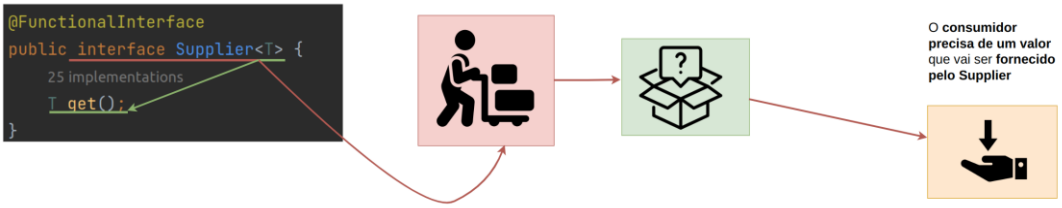
Como as **streams** conseguem ser encadeadas uma após a outra? Pelo uso das **Interfaces funcionais**, que são basicamente uma **Interface que tem um único método abstrato**, pra facilitar o uso de expressões lambdas.

Toda **expressão lambda que você faz no encadeamento de streams,** é uma **classe anônima que implementa a expressão de acordo com a Interface Funcional** esperada como parâmetro.



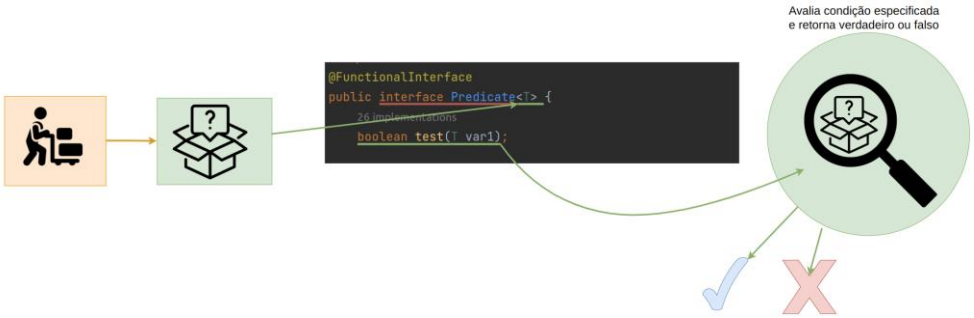
Principais Interfaces Funcionais – Java Funcional

Supplier: É um tipo de interface funcional que não recebe nenhum parâmetro, mas retorna algo. A ideia dela é realmente ser um ***"fornecedor"***, o ***supplier*** quer entregar para **alguém algum valor *para ser consumido:***

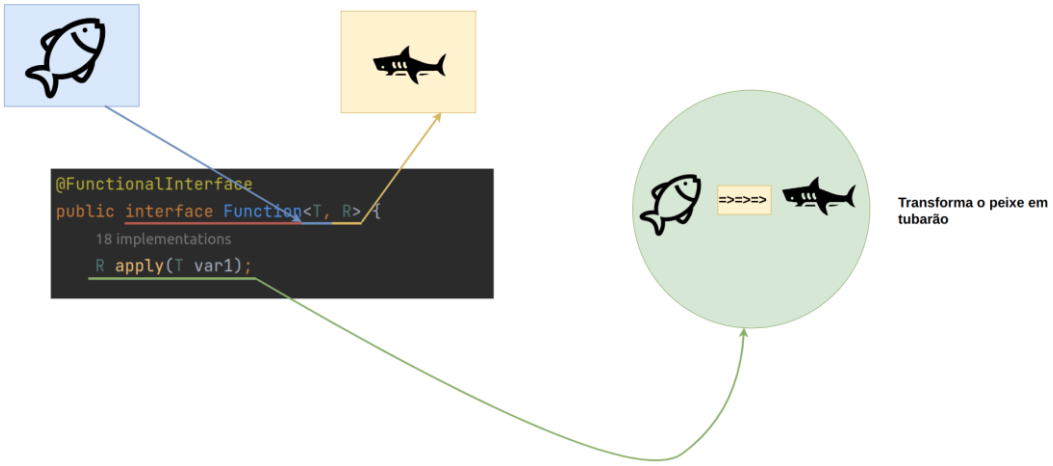


Consummer: O ***consumidor*** é o oposto do Supplier, ele não retorna nada, mas consome algum valor. A ideia dele é simplesmente ***fazer algo com o valor recebido*** e não ***devolver nada***, é como se fosse uma ***operação final***, tipo o ***forEach,*** ele vai receber um valor, fazer algo e já era:

Predicate: Interface funcional com um ***método de teste***, você pode usar o **Predicate<T>** para criar regras em cima de um determinado objeto, seja para usar em filtros ou validar regras de negócio, ela é flexível o suficiente para ser usada em vários cenários de teste ***(equals, not, or)***:



Function : Representa a composição mais básica de uma função, ou seja, ela **recebe alguma coisa** e **devolve outra**. O **tipo que a Function recebe pode ser diferente do tipo que ela devolve**, esse conceito é bem visto em **"map"**:



Bifunction: pode receber dois tipos, mas devolve um **| BiPredicate:** recebe dois tipos, mas devolve um boolean apenas **| BiConsummer :** consome dois tipos, mas não retorna nada.

Interfaces Funcionais de Operação – Java Funcional

As ***interfaces de operação (UnaryOperator e BynariOperator)*** estendem de ***Function e BiFunction***, ambas são funções, mas a diferença primordial é que **eles recebem e devolvem O MESMO TIPO**. Essas **Interfaces** usam o **método abstrato das Interfaces pai**.

Dá pra entender essas **Interfaces como uma restrição** pra que uma **função mantenha o comportamento esperado em uma operação de reduce,** por exemplo, onde eu **tenho que garantir que estou somando dois argumentos DO MESMO TIPO.**