



High performance. Delivered.

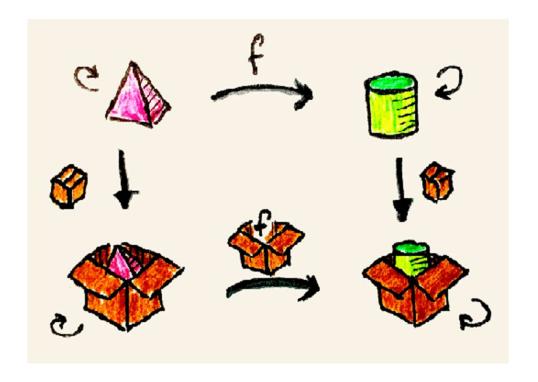
Application Delivery Fundamentals: Java

Module 12: Programação Funcional (Lambda e Streams)

"PROGRAMAÇÃO FUNCIONAL"

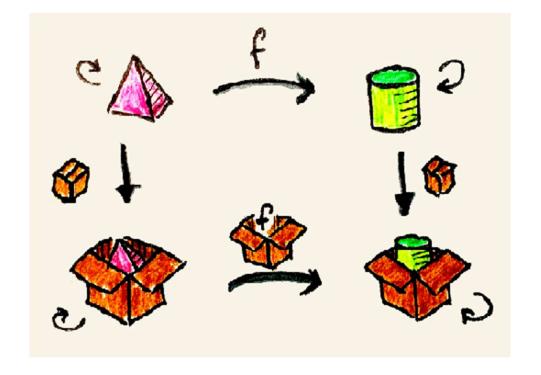
Programação Funcional: É um paradigma de programação;

 No paradigma funcional eu não digo ao meu código o que ele deve fazer, quando e como. Não irei desenvolvê-lo passo a passo. Eu penso meu código como uma sequência de funções e/ou passos, as quais de maneira composta irão resolver meu problema.



Programação Funcional: É um paradigma de programação;

- Pense da seguinte forma: eu tenho um dado de entrada e preciso transformá-lo em um dado de saída.
- Usando PF eu vou abstrair as lógicas de transformações do meu código em funções, e usálas no momento oportuno para transformar este meu dado.



"LAMBDA"

O que é LAMBDA em java?

- A expressão lambda é tratada como uma função, portanto, o compilador não cria o arquivo .class.
- Habilite para tratar a funcionalidade como um argumento de método ou o código como dados.
- Uma função que pode ser criada sem pertencer a nenhuma classe.
- Uma expressão lambda pode ser transmitida como se fosse um objeto e executada sob demanda;

Por que Usar Lambda?

- Para fornecer a implementação da interface funcional.
- Menos codificação.

Sintaxe



Zero parameter:

```
() -> System.out.println("Zero parameter lambda");
```

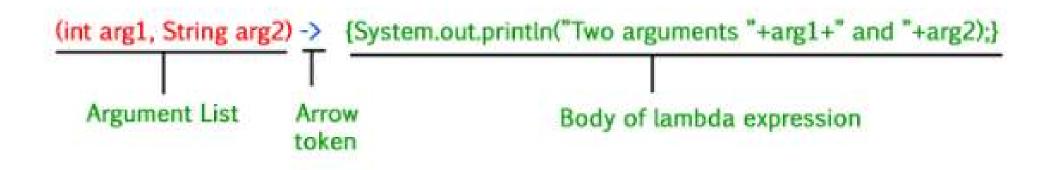
One parameter:-

```
(p) -> System.out.println("One parameter: " + p);
```

Multiple parameters :

```
(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2
```

Sintaxe



Arrays.sort(dogArray, new Comparator<Dog>() {
 @Override
 public int compare(Dog o1, Dog o2) {
 return o1.getWeight() - o2.getWeight();
 }
 });

Lambda Expression

Arrays.sort(dogArray, (m, n) -> m.getWeight() - n.getWeight());

- 1. Uma lista separada por vírgula de parâmetros formais entre parênteses. Nesse caso, é (Dog m, Dog n)
- 2. O token de seta ->
- 3. Um corpo, que consiste em uma única expressão ou em um bloco de instruções.

Nesse caso, é uma expressão única - Integer.compare (m.getWeight (), n.getWeight ())

ForEach

Sintaxe

```
List<Integer> intSeq = Arrays.asList(1,2,3);
intSeq.forEach(x -> {
  int y = x * 2;
  System.out.println(y);
});
```

"CLOSURE"

Closures são expressões com valor de função embutida, o que significa que são funções de classe com variáveis limitadas.

Closures podem ser passados para outra função como um parâmetro.

Um closure nos dá acesso à função externa de uma função interna.

Closure

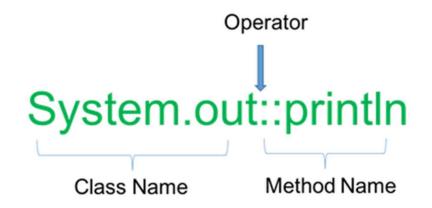
```
@FunctionalInterface // optional
public interface NumToTextConverter {
    String convert(int x);
static void closureType1Demo() {
    NumToTextConverter textOfWeekday = new NumToTextConverter() {
        String [] weeks = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
        @Override
        public String convert(int num) {
            return (num > 0 && num <= weeks.length) ? weeks[num-1] : null;
        };
    System.out.println(textOfWeekday.convert(1)); // Mon
```

Closure

```
1 package acc.lambda.sample;
 3 public class LambdaZero {
          @FunctionalInterface
 40
          interface Saudacoes {
             void digaAI(String message);
          public static void main(String args[]) {
 9⊝
10
             //Sem parentesis
11
              Saudacoes saudacoes1 = message -> System.out.println("Ola " + message);
12
13
             //Com parentesis
14
              Saudacoes saudacoes2 = (message) -> System.out.println("Hello " + message);
15
16
             saudacoes1.digaAI("Maria");
17
18
             saudacoes2.digaAI("Jhon Doe");
```

"REFERENCE METHOD"

Sintaxe da chamada do método por referência



array.forEach(System.out::println);

MethodReferencesExamples1,2 e 3.java

Lambda Expression vs Method Reference

Method Reference
String :: toString
String :: toLowerCase
String :: length
Integer :: compareTo
String :: compareTo

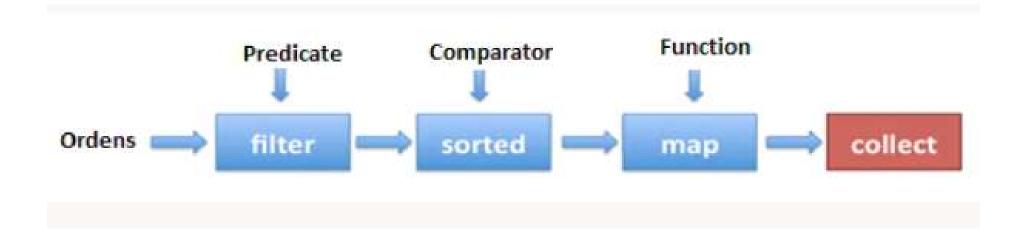
LambdaVSmethod.java
MapCollectStream.java

"STREAM"



O que é Stream Processing?

- ou Processamento de Fluxo em tradução livre;
- Dado um conjunto de dados (um fluxo ou stream), uma série de operações (funções do <u>kernel</u>) é aplicada a cada elemento na corrente, ou seja, um streaming uniforme, onde uma operação é aplicada a todos os elementos do fluxo.

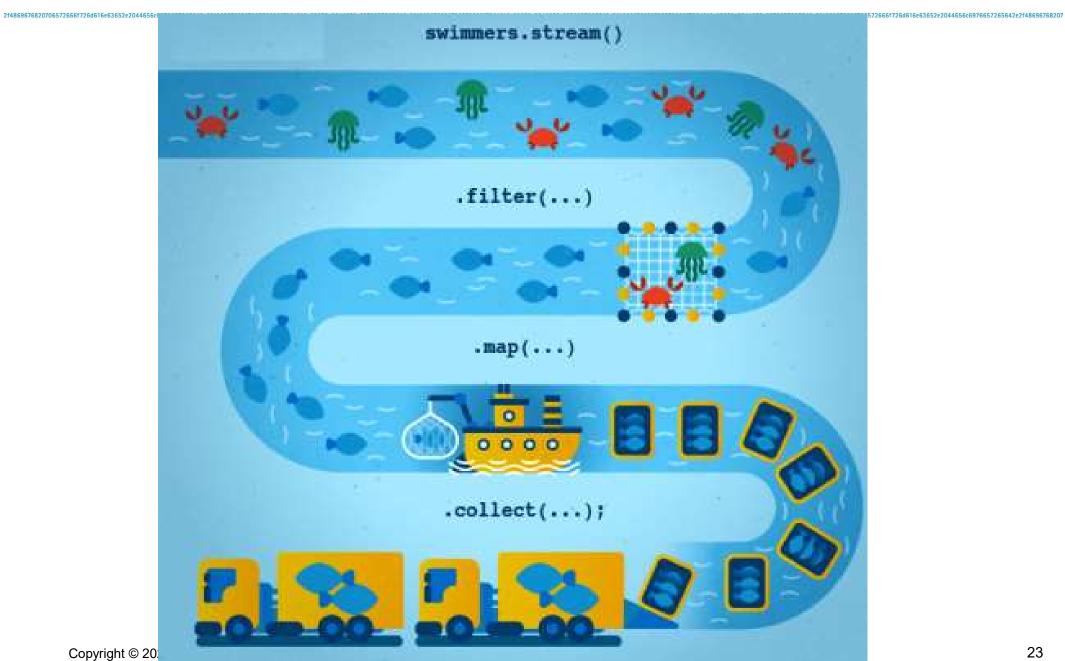


O que é Stream Processing?

- Sequência de elementos de uma fonte de dados que suporta operações de agregação;
- Sequência de elementos: Uma stream oferece uma interface para um conjunto de valores sequenciais de um tipo de elemento particular. Apesar disso, as streams não armazenam elementos; estes são calculados sob demanda.
- Fonte de dados: As streams tomam seu input de uma fonte de dados, como coleções, matrizes ou recursos de E/S.
- Operações de agregação: As streams suportam operações do tipo SQL e operações comuns à maioria das linguagens de programação funcionais, como filter, map, reduce, find, match e sorted, entre outras.

O que é Stream?

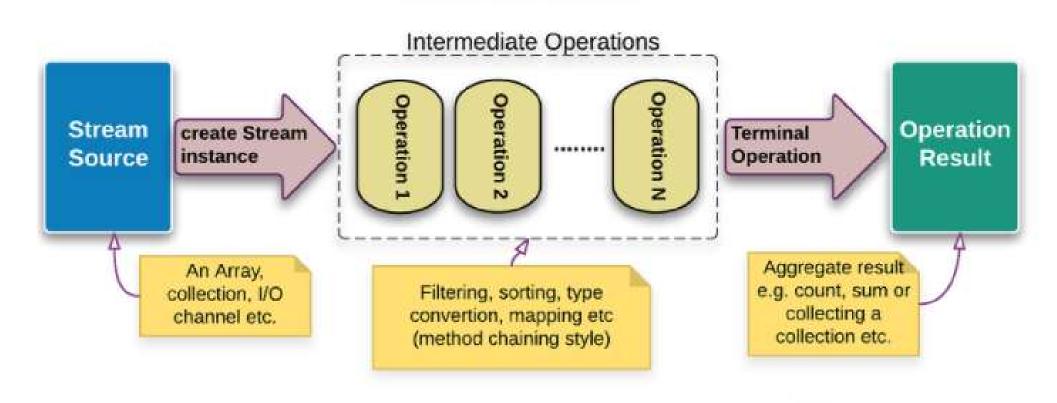
- A Streams API traz uma nova opção para a manipulação de coleções em Java seguindo os princípios da programação funcional.
- Combinada com as expressões lambda, ela proporciona uma forma diferente de lidar com conjuntos de elementos, oferecendo ao desenvolvedor uma maneira simples e concisa de escrever código que resulta em facilidade de manutenção e paralelização sem efeitos indesejados em tempo de execução.
- Com o aperfeiçoamento constante do hardware, sobretudo a proliferação das CPUs multicore, a API levou isso em consideração e com o apoio do paradigma funcional, suporta a paralelização de operações para processar os dados – abstraindo a lógica de baixo nível para se ter um código multithreading – e
- Deixa o desenvolvedor concentrar-se totalmente nas regras de negócio.



23

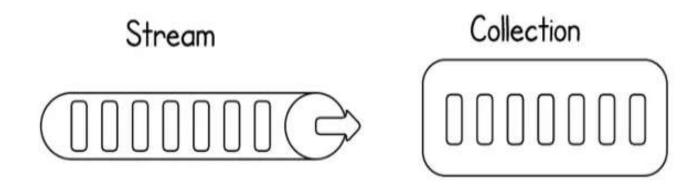
Copyright © 20

Java Streams



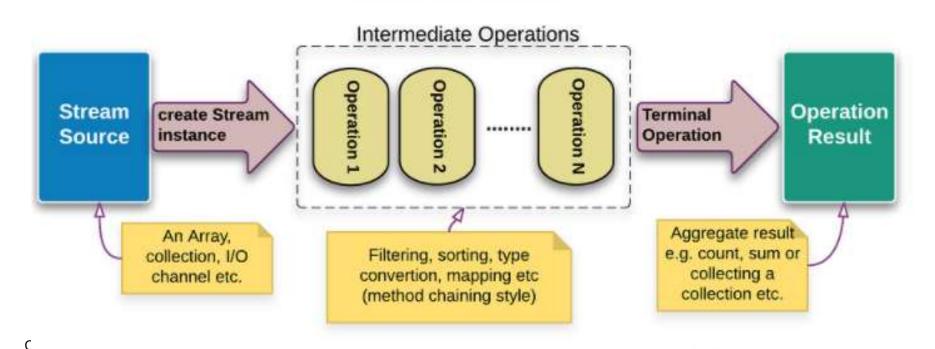
STREAM vs Collection

- As collection se referem a dados enquanto as stream se referem a cálculos:
- java.util.stream.Stream



- Quando usa-se streams, geralmente trabalhamos com três elementos:
- fonte de dados (p. ex., uma coleção) sobre a qual vai se fazer uma consulta;
- cadeia de operações intermediárias, que formam um processo;
- operação terminal, que executa o processo da stream e produz um resultado.

Java Streams



• Filtragem:

- filter(Predicate): Retorna uma stream incluindo todos os elementos que coincidem com o predicado indicado.
- distinct: Retorna uma stream com elementos únicos;
- limit(n): Retorna uma stream cuja tamanho máximo é n.
- skip(n): Retorna uma stream em que se descartaram os primeiros n elementos.

Pesquisas e identificação de coincidências:

- Um padrão comum no processamento de dados é determinar se alguns elementos se ajustam a uma propriedade determinada.
- É possível usar as operações anyMatch, allMatch e noneMatch para esse fim.
- Todas elas tomam como argumento um predicado e retornam um valor boolean (isto é, são operações terminais).
- Por exemplo, podemos usar allMatch para verificar que todos os elementos de uma stream de preços tenham valores superiores a 100.
- boolean despesa = preco.stream().allMatch(t -> t.getValue() > 100);

findFirst e findAny:

- Para recuperar elementos arbitrários de uma stream;
- Podem ser usadas juntamente com outras operações de stream, como filter.
- Tanto findFirst como findAny retornam um objeto Optional

Optional<T> (java.util.Optional)

- É uma classe que representa a existência ou ausência de um valor;
- A classe Optional contém diversos métodos para comprovar a existência de um elemento.
- •
- Tanto findFirst como findAny retornam um objeto Optional

Mapeamento

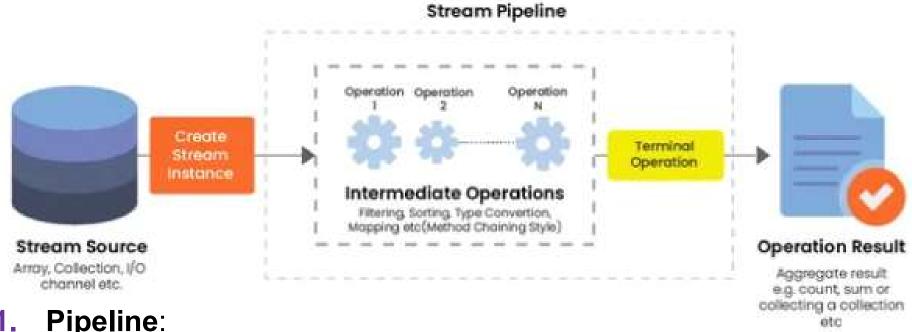
- Projeta os elementos da stream em outro formato;
- A função é aplicada a cada elemento, que é "mapeado".

```
List<String> words = Arrays.asList("Oracle", "Java", "Magazine");
  List<Integer> wordLengths = words.stream()
.map(String::length)
.collect(toList());
```

Redução

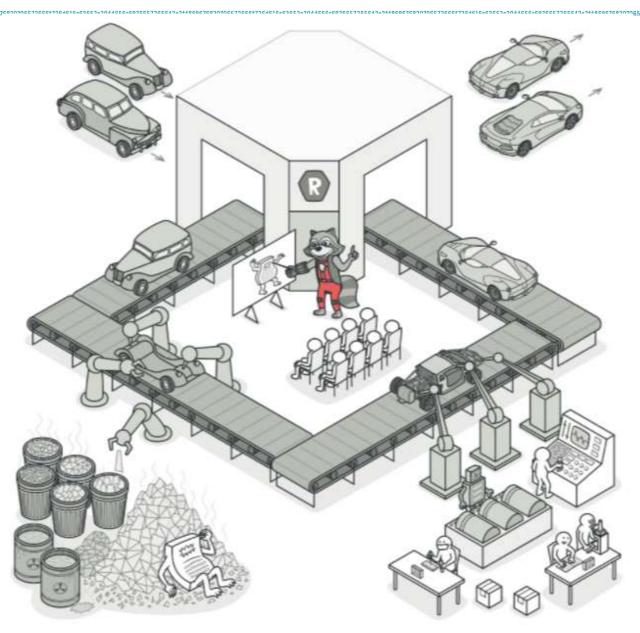
- Usamos collect para combinar todo o conjunto de elementos de uma Stream em um objeto List.
- (ex. StreamCollect.java)

Java Streams

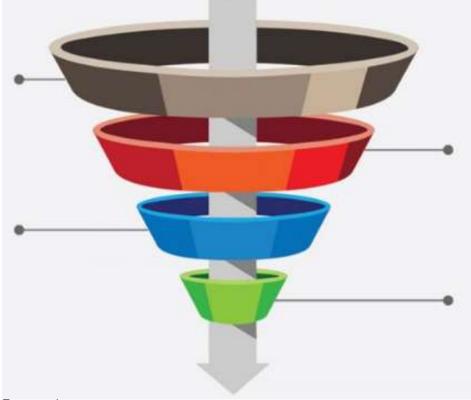


- Pipeline:
- É o processo pelo qual uma instrução de processamento é subdividido em etapas;
- É utilizada para acelerar a velocidade de operação da CPU;

PipeLine

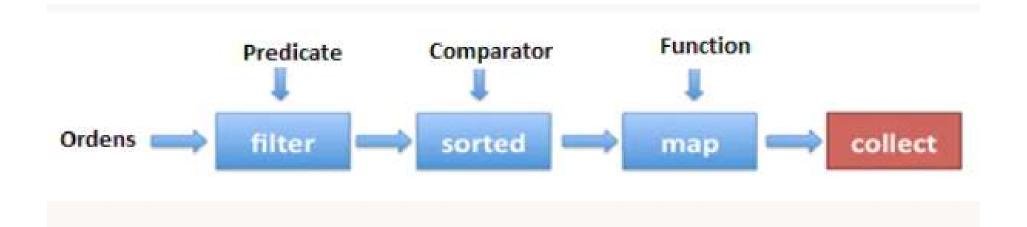


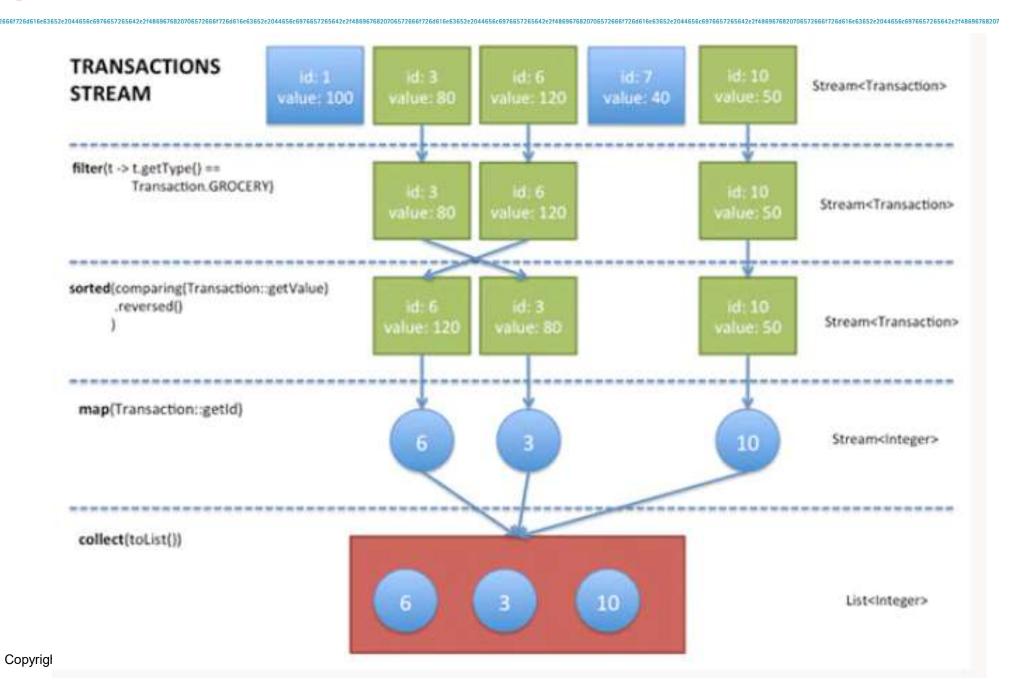
- 1. Pipeline:
- 2. É o processo pelo qual uma instrução de processamento é subdividido em etapas;
- 3. É utilizada para acelerar a velocidade de operação da CPU;
- 4. Viabiliza a agregação de dados;
- 5. Dados agregados são essencialmente dados sumarizados.;

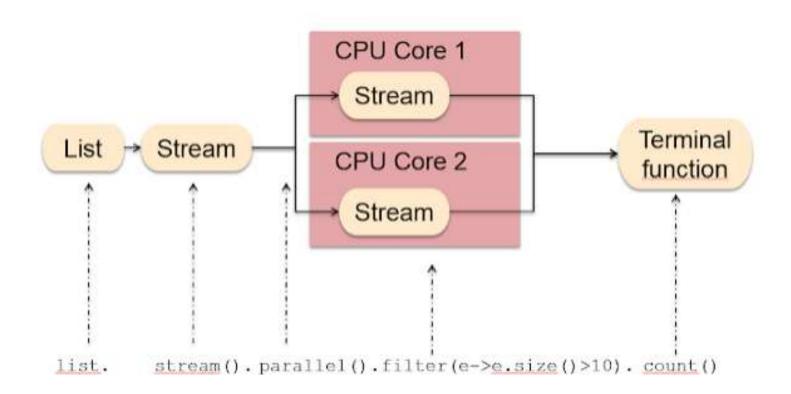


```
List<Integer> ordensIDs = Ordem.stream()
    .filter(o -> o.getType() == Ordem.ATIVACAO)
    .sorted(comparing(Ordem::getValue).reversed())
    .map(Ordem::getId)
    .collect(toList());
```

2468696768207065726661726661665765365262044656c697665726564262748696768707065726661726661666365262044656c697665726564262748696768207065726661726661666365262044656c697665726564262748696768207065726661726677266617266617266617266617266772666172666172666772666172667726667726667726667726667726667726667726667726667726667726667726667726772667726677267726772667726677267726772677







https://www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/stream-cheat-sheet.html

Perguntas:

O que é Programação Funcional?

Programar baseado em funções;

O que é Lambda?

Uma função seta que pode ser criada sem pertencer a nenhuma classe;

O que é Stream?

Um stream é a maneira de abstrair e especificar como processar uma agregação

Questions and Comments

 What questions or comments do you have?



