Visão Geral de *Streams*

Prof. Jeferson Souza, MSc. (thejefecomp)

jeferson.souza@udesc.br

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

JOINVILLE
CENTRO DE CIÊNCIAS
TECNOLÓGICAS

Introdução às Streams

Afinal, o que são as famosas Streams [Boyarsky&Selikoff, 2016] [Oracle, 2021]?

Streams são sequências de elementos/dados nas quais operações podem ser aplicadas de forma sequencial ou paralela, a terminar na produção de um resultado esperado.

Introdução às Streams

Principais características das Streams...

- Permitem a organização e composição de operações em uma estrutura chamada *Stream Pipeline*;
- Permitem a execução de operações de forma sequencial ou paralela;
- São preguiçosas, a implicar na execução de operações somente quando necessário;
- Não possibilitam acesso direto e/ou manipulação de seus elementos;
- Implementações podem otimizar a produção dos resultados, caso necessário.

Introdução às Streams

Definição de Stream Pipeline [Oracle, 2021]

Um **Stream Pipeline** consiste em uma fonte de dados (que teoricamente pode ser infinita), um conjunto finito de operações intermediárias, a permitir a transformação de uma *Stream* em outra, e uma operação terminal para produzir o resultado esperado.

Pense em um Stream Pipeline Como...

Uma linha de produção em um chão de fábrica, onde cada operação representa um estágio da referida linha, da entrada até a produção do resultado final.

Introdução às Streams

Exemplo de uso das Streams [Oracle, 2021]

```
int soma = widgets.stream()
                  .filter(w → w.getCor() == VERMELHO)
                  .mapToInt(w → w.getPeso())
                  .sum();
```

No exemplo acima temos especificado o processamento de uma coleção de *widgets*, onde está-se interessado em somar o peso de cada um dos *widgets* a possuir a cor vermelha. Portanto, o *pipeline* para a dada *Stream* é estabelecido pela coleção de entrada *widgets*, as operações intermediárias *filter()* e *mapToInt()*, e a operação terminal *sum()*.

Aplicação de Programação Funcional em uma Linguagem Orientada a Objetos

Vamos deixar as coisas mais declarativas [Boyarsky&Selikoff, 2016]...

O advento da possibilidade do uso de uma forma mais declarativa na escrita de código-fonte foi introduzida na versão 8 da linguagem Java, a implicar na aplicação de técnicas de programação funcional em uma linguagem puramente orientada a objetos. Bem vindo ao mundo das expressões Lambda :-D.

Aplicação de Programação Funcional em uma Linguagem Orientada a Objetos

Expressões Lambda em Java [Boyarsky&Selikoff, 2016]...

As expressões *Lambda* são definidas em Java como blocos de código-fonte que permitem a especificação de um conjunto de instruções de forma declarativa, a implicar em uma maior expressividade aquando combinadas com o poder da orientação a objetos presente na linguagem Java. *Interfaces* funcionais, as quais contém somente um único método abstrato, são a base para a definição e o uso das expressões *Lambda*. A interface *java.util.function.Predicate* é um exemplo de interface funcional.

Aplicação de Programação Funcional em uma Linguagem Orientada a Objetos

Expressões Lambda em Java [Boyarsky&Selikoff, 2016]...

As expressões *Lambda* estão divididas em três partes principais:

- Lista de parâmetros: especifica os parâmetros utilizados dentro do corpo da expressão;
- Seta ($\rightarrow$): especifica a associação declarativa entre os parâmetros definidos do lado esquerdo, e o trecho de código a ser executado no lado direto (i.e. o corpo da expressão);
- Corpo: trecho de código-fonte a ser executado aquando da avaliação da expressão *Lambda* especificada.

Aplicação de Programação Funcional em uma Linguagem Orientada a Objetos

Exemplos de Expressões Lambda em Java...

1. **fruta → System.out.println(fruta)**
2. **(Fruta fruta) → { System.out.println(fruta); }**
3. **fruta → fruta.ehComestivel()**
4. **(Fruta fruta) → { return fruta.ehComestivel(); }**
5. **() → new Fruta()**
6. **() → { return new Fruta(); }**
7. **(fruta,equipamento) → equipamento.processar(fruta)**

A Interface Funcional Predicate

A interface funcional Predicate [Oracle, 2021]

A interface funcional *Predicate* representa um predicado de um único argumento, a ser especificado por um método de retorno boleano chamado *test()*, a permitir verificar a validade do predicado quando aplicado a um dado argumento.

```
@FunctionalInterface
public interface Predicate<T>{
  ... A omitir todos os métodos default e estáticos ...
  boolean test(T t);
}
```

A Interface Funcional Predicate

Exemplo de uso da interface funcional Predicate

```
List<String> lCarros = new ArrayList<>();
lCarros.add("Corsa");// Lista → [Corsa]
lCarros.add("Pálio");// Lista → [Corsa,Pálio]
lCarros.add("Chevete");// Lista → [Corsa,Pálio,Chevete]
lCarros.add("Fusca");// Lista → [Corsa,Pálio,Chevete,Fusca]
Predicate<String> comecaComC = carro →
carro.startsWith("C");
lCarros.removeIf(comecaComC);// Lista → [Pálio,Fusca]
```

Bibliografia

GOSLING, J.; JOY, B.; STEELE, G.; BRACHA, G.; BUCKLEY, A.; SMITH, D.; BIERMAN, G. **"The Java Language Specification: Java SE 16 Edition"**. Oracle. 2021. Disponível em: `https://docs.oracle.com/javase/specs/jls/se16/jls16.pdf`. Acesso em: 12 Jul. 2021.

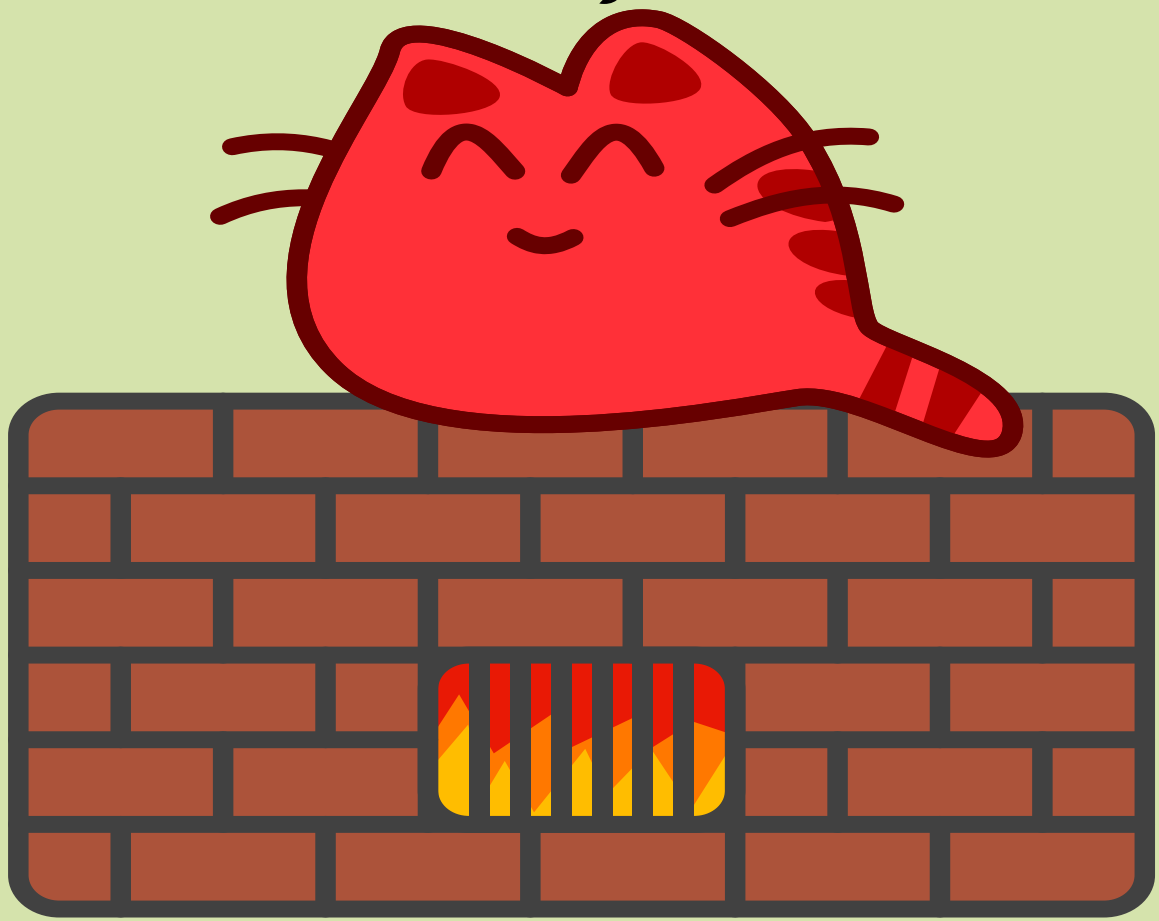
ORACLE, INC. **"Java Platform, Standard Edition & Java Development Kit Version 16 API Specification"**. Oracle. 2021. Disponível em: `https://docs.oracle.com/en/java/javase/16/docs/api/index.html`. Acesso em: 26 Jul. 2021.

Bibliografia (Continuação)

BOYARSKY, J. and SELIKOFF, S. **"Oracle Certified Associate Java SE 8 Programmer I: Study Guide"**. Sybex. Indianapolis, Indiana, USA. 2015.

BOYARSKY, J. and SELIKOFF, S. **"Oracle Certified Associate Java SE 8 Programmer II: Study Guide"**. Sybex. Indianapolis, Indiana, USA. 2016.

That's it folks!



Thank you for your attention!