Java Funcional

Leonardo Mendes gomes dos Santos

Este vídeo pode ser utilizado nas aulas do Centro Paula Souza

Programação Funcional

- Programação funcional é um paradigma de programação;
- Enfatiza o uso de funções e evita mudanças de estado ou dados mutáveis;

Programação Funcional

- **Vantagens x Desvantagens**

– Fácil manutenção;

– Facilidade para processamento em paralelo;

– Facilidade nos testes e na busca por bugs;

– Caminho para se pensar de forma funcional é mais complexo para quem já programou em linguagens imperativas.

Paradigmas de programação

- Imperativo (C, Pascal, Fortran, Cobol)
- Orientado a objetos (C++, Object Pascal, Java (< 8), C# (< 3))
- Funcional (Haskell, Closure, Clean, Erlang)
- Lógico (Prolog)
- Multiparadigma (JavaScript, Java (8+), C# (3+), Ruby, Python, Go)

Interface Funcional

- É uma interface que possui um único método abstrato. Suas implementações serão tratadas como expressões lambda.

```
public class MyComparator implements Comparator<Product>{
    @Override
    public int compare(Product o1, Product o2) {
        return o1.getName().toUpperCase().compareTo(o2.getName().toUpperCase());
    }
}
public static void main(String[] args) {
    (...)
    list.sort(new MyComparator());
}
```

https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html#:~:text=function%20Description,types%20are%20matched%20or%20adapted.

Lambda Expression

"também conhecida como arrow function"

- (parâmetros) -> { Corpo da função};

Exemplos:

- (a,b) -> a != b; "Quando a função tem apenas uma linha não é preciso colocar chaves";
- x -> x + 2; "Quando se passa um único parâmetro não se precisa de Parênteses";

As interfaces funcionais mais comuns

- Predicate
  https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html

- Function
  https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html

- Consumer
  https://docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html

Method References

- Permite referenciar métodos ou construtores usando ::
- Como se fosse uma abreviação de uma lambda, chamando somente um método

Lambda:

```
Consumer<String> cons = (String s) -> System.out.println(s);
```

Method Reference:

```
Consumer<String> cons = System.out::println;
```

Stream

- O Stream é um novo jeito de se interar sobre coleções.

- É uma sequencia de elementos advinda de uma fonte de dados que oferece suporte a "operações agregadas".

  Fonte de dados: coleção, array, função de iteração, recurso de E/S

- “Você pode enxergar o Stream como uma sequencia de dados e para cada objeto ele aplica uma função”

Examplo:

```
List<Product> expensiveProducts = products.stream().filter(x -> x.getPrice() > 6000).collect(Collectors.toList);
```

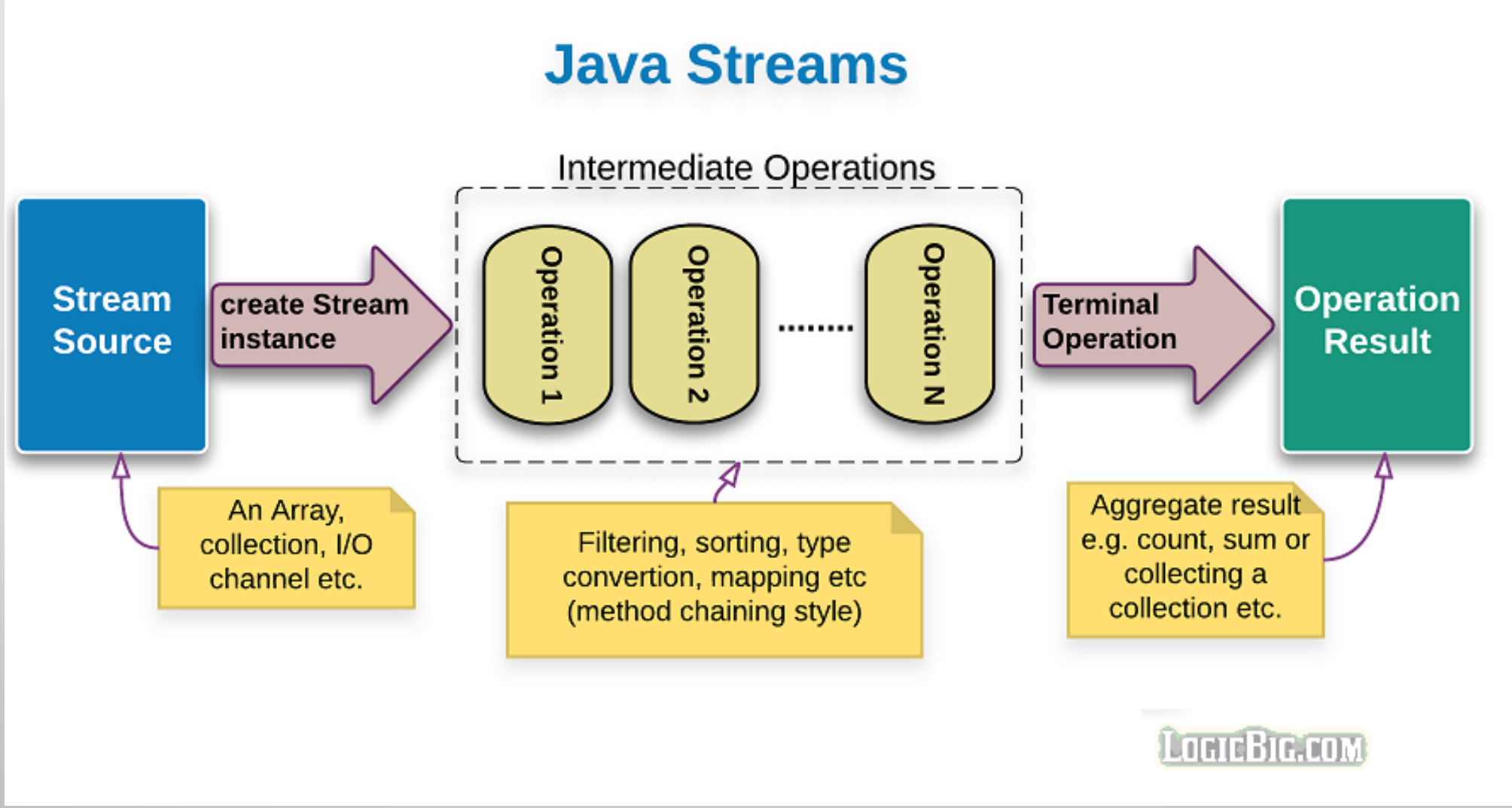
```
List<String> list = products.stream().map(p -> p.getName().toUpperCase()).collect(Collectors.toList());
```

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

Caracteristicas do Stream

- A iteração acontece de forma escondida do programador.
- Thread safe
- Sem efeitos colaterais
- Os dados de uma Stream serão consumidos somente quando necessário
- Não a índices
- O Stream só pode ser usado uma vez.
- Cada operação com stream gera uma nova stream.

Stream Pipeline



Operações intermediárias e terminais

- O pipeline é composto por zero ou mais operações intermediárias e uma terminal.
- Operação intermediária:

  Produz uma nova streams (encadeamento)
- Só executa quando uma operação terminal é invocada (lazy evaluation)
- Operação terminal:

  Produz um objeto não-stream (coleção ou outro)
- Determina o fim do processamento da stream

Operações Intermediarias

- filter
- map
- flatmap
- peek
- distinct
- sorted
- skip
- limit (*)

Operações Terminais

- forEach
- forEachOrdered
- toArray
- reduce
- collect
- min
- max
- count
- anyMatch (*)
- allMatch (*)
- noneMatch (*)
- findFirst (*)
- findAny (*)

Referências:

Curso “Java COMPLETO 2020 Programação Orientada a Objetos + Projetos” do Nélio Alves.