PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
DEPARTAMENTO DE COMPUTAÇÃO
TÉCNICAS DE PROGRAMAÇÃO 1 – CMP 1046
PROF. MSC. ANIBAL SANTOS JUKEMURA

[Sintaxe Lambda]

PUC GOIÁS

Agenda:

- Dica: métodos default em interfaces
- Introdução
- Iteração externa x iteração interna
- Interfaces funcionais
- Expressões Lambda
- Fluxos

PUC GOIÁS

Métodos Default em Interfaces

Exemplo em Sala de Aula

Sintaxe Lambda - Introdução

- Antes do Java SE 8, o Java suportava três paradigmas de programação
- programação procedural
- programação orientada a objetos
- programação genérica.

- O Java SE 8 acrescenta **a programação funcional**. As novas capacidades da linguagem e biblioteca que suportam esse paradigma foram adicionadas ao Java como parte do Projeto Lambda:

http://openjdk.java.net/projects/lambda

PUC GOIÁS

Sintaxe Lambda - Iteração externa x iteração interna

int[] values = {3, 10, 6, 1, 4, 8, 2, 5, 9, 7};

```
int sum = 0;
for (int counter = 0; counter < values.length; counter++)
   sum += values[counter];
```

Iteração Externa

Existem **várias possibilidades de erros**. Por exemplo, você pode inicializar a variável sum incorretamente, inicializar a variável de controle counter incorretamente, usar a condição de continuação de loop errada, incrementar a variável de controle counter de modo incorreto ou adicionar incorretamente cada valor no array a sum.

```
System.out.printf("%nSoma dos inteiros de 0 a 9: %d%n", IntStream.of(values).sum());
```

Iteração Interna

PUC GOIÁS

Sintaxe Lambda - Iteração externa x iteração interna

Iteração interna

- Na programação funcional, você especifica o que quer alcançar em uma tarefa, mas não como alcançar isso. Você não precisa especificar como iterar pelos elementos ou declarar e usar quaisquer variáveis mutáveis. Isso é conhecido como repetição interna, porque a biblioteca determina como acessar todos os elementos para realizar a tarefa.

- Com a iteração interna, você pode facilmente informar a biblioteca que você deseja realizar essa tarefa com o processamento paralelo para tirar proveito da arquitetura multiprocessada do seu computador — isso pode melhorar significativamente o desempenho da tarefa.

- As capacidades da programação funcional focalizam a imutabilidade — não modificar a origem de dados que é processada ou qualquer outro estado do programa.

PUC GOIÁS

Sintaxe Lambda - Interfaces funcionais

Interface	Descrição
`BinaryOperator<T>`	Contém o método `apply`, que recebe dois argumentos `T`, realiza uma operação neles (como um cálculo) e retorna um valor do tipo `T`. Veremos vários exemplos de `BinaryOperators` a partir da Seção 17.3.
`Consumer<T>`	Contém o método `accept`, que recebe um argumento `T` e retorna `void`. Realiza uma tarefa com o argumento `T`, como gerar uma saída do objeto, chamar um método do objeto etc. Veremos vários exemplos de `Consumers` a partir da Seção 17.3.
`Function<T,R>`	Contém o método `apply`, que recebe um argumento `T` e retorna um valor do tipo `R`. Chama um método no argumento `T` e retorna o resultado desse método. Veremos vários exemplos de `Functions` a partir da Seção 17.5.
`Predicate<T>`	Contém o método `test`, que recebe um argumento `T` e retorna um `boolean`. Testa se o argumento `T` atende uma condição. Veremos vários exemplos de `Predicates` a partir da Seção 17.3.
`Supplier<T>`	Contém o método `get`, que não recebe argumentos e produz um valor do tipo `T`. Muitas vezes usado para criar um objeto de coleção em que os resultados de uma operação de fluxo são inseridos. Veremos vários exemplos de `Suppliers` a partir da Seção 17.7.
`UnaryOperator<T>`	Contém o método `get` que não recebe argumentos e retorna um valor do tipo `T`. Veremos vários exemplos de `UnaryOperators` a partir da Seção 17.3.

Figura 17.2 | As seis interfaces funcionais genéricas básicas no pacote `java.util.function`.

T e R são nomes de tipo genérico que representam o tipo do objeto no qual a interface funcional opera e o tipo de retorno de um método, respectivamente.
Há muitas outras interfaces funcionais no pacote java.util.function que são versões especializadas daquelas na Figura 17.2.

PUC GOIÁS

Sintaxe Lambda - Fluxos

Operações Stream intermediárias	
`filter`	Resulta em um fluxo contendo apenas os elementos que atendem uma condição.
`distinct`	Resulta em um fluxo que contém somente os elementos únicos.
`limit`	Resulta em um fluxo com o número especificado de elementos a partir do início do fluxo original.
`map`	Resulta em um fluxo em que cada elemento do fluxo original é mapeado para um novo valor (possivelmente de um tipo diferente) — por exemplo, mapear valores numéricos para as raízes quadradas dos valores numéricos. O novo fluxo tem o mesmo número de elementos que o fluxo original.
`sorted`	Resulta em um fluxo em que os elementos estão em ordem classificada. O novo fluxo tem o mesmo número de elementos que o fluxo original.

Figura 17.3 | Operações `Stream` intermediárias comuns.

PUC GOIÁS

Sintaxe Lambda - Fluxos

Lambda	Descrição
`String::toUpperCase`	Referência de método para um método de instância de uma classe. Cria uma lambda de um parâmetro que chama o método de instância sobre o argumento da lambda e retorna o resultado do método. Usada na Figura 17.7.
`System.out::println`	Referência de método para um método de instância que deve ser chamado em um objeto específico. Cria uma lambda de um parâmetro que chama o método de instância sobre o objeto especificado — passando o argumento da lambda para o método de instância — e retorna o resultado do método. Usada na Figura 17.10.
`Math::sqrt`	Referência de método para um método `static` de uma classe. Cria uma lambda de um parâmetro em que o argumento da lambda é passado para o método `static` especificado e a lambda retorna o resultado do método.
`TreeMap::new`	Referência de construtor. Cria uma lambda que chama o construtor sem argumentos da classe especificada para criar e inicializar um novo objeto dessa classe. Usada na Figura 17.17.

Figura 17.8 | Tipos de referências de método.

PUC GOIÁS

Sintaxe Lambda - Expressões Lambda

- A programação funcional é realizada com **expressões lambda**.
- Uma **expressão lambda** representa um método anônimo — a notação abreviada para implementar uma interface funcional.
- O tipo de uma expressão lambda é o tipo da interface funcional que a expressão lambda implementa.
- Expressões lambda podem ser usadas em qualquer lugar em que interfaces funcionais são esperadas.

PUC GOIÁS

Sintaxe Lambda - Expressões Lambda

Sintaxe lambda

Uma lambda consiste em uma *lista de parâmetros* seguida pelo símbolo de seta (->) e um corpo, como em:

```
(listaDeParâmetros) -> {instruções}
```

A seguinte lambda recebe dois `ints` e retorna sua soma:

```
(int x, int y) -> {return x + y;}
```

Quando o corpo contém uma única expressão, a palavra-chave return e as chaves { } podem ser omitidas

```
(x, y) -> x + y
```

Quando a lista de parâmetros contém um único parâmetro, os parênteses podem ser omitidos

```
value -> System.out.printf("%d ", value)
```

Para definir uma lambda com uma lista de parâmetros vazia

```
() -> System.out.println("Welcome to lambdas!")
```

Sintaxe Lambda - Fluxos

- O Java SE 8 introduz o conceito de **streams**, que são semelhantes aos **iteradores (iterators).**

- Fluxos são objetos das classes que implementam a interface Stream (do pacote **java.util.stream**) ou uma das interfaces de fluxo especializadas para processar coleções de valores **int**, **long** ou **double** .

- Juntamente com lambdas, fluxos permitem realizar tarefas sobre coleções de elementos, muitas vezes a partir de um objeto **array** ou **coleção**.

PUC GOIÁS

Sintaxe Lambda - Fluxos

Exemplo 1 em Sala de Aula

`forEach`	Realiza o processamento em cada elemento em um fluxo (por exemplo, exibir cada elemento).
`count`	Retorna o *número de elementos* no fluxo.
`max`	Localiza o *maior* valor em um fluxo numérico.
`min`	Localiza o *menor* valor em um fluxo numérico.
`average`	Calcula a *média* dos elementos em um fluxo numérico.
`reduce`	Reduz os elementos de uma coleção a um *único valor* usando uma função de acumulação associativa (por exemplo, uma lambda que adiciona dois elementos).
`filter`	Resulta em um fluxo contendo apenas os elementos que atendem uma condição.
`sorted`	Resulta em um fluxo em que os elementos estão em ordem classificada. O novo fluxo tem o mesmo número de elementos que o fluxo original.
`map`	Resulta em um fluxo em que cada elemento do fluxo original é mapeado para um novo valor (possivelmente de um tipo diferente) — por exemplo, mapear valores numéricos para as raízes quadradas dos valores numéricos. O novo fluxo tem o mesmo número de elementos que o fluxo original.
`findFirst`	Localiza o *primeiro* elemento no fluxo com base nas operações intermediárias anteriores; termina imediatamente o processamento do pipeline do fluxo depois que esse elemento é encontrado.

PUC GOIÁS

Sintaxe Lambda - Fluxos

Exemplo 2 em Sala de Aula

`collect`	Cria uma *nova coleção* dos elementos contendo os resultados das operações anteriores do fluxo.
`filter`	Resulta em um fluxo contendo apenas os elementos que atendem uma condição.
`sorted`	Resulta em um fluxo em que os elementos estão em ordem classificada. O novo fluxo tem o mesmo número de elementos que o fluxo original.

PUC GOIÁS

Sintaxe Lambda - Expressões Lambda

Exemplo 3 em Sala de Aula

`sorted`	Resulta em um fluxo em que os elementos estão em ordem classificada. O novo fluxo tem o mesmo número de elementos que o fluxo original.
`map`	Resulta em um fluxo em que cada elemento do fluxo original é mapeado para um novo valor (possivelmente de um tipo diferente) — por exemplo, mapear valores numéricos para as raízes quadradas dos valores numéricos. O novo fluxo tem o mesmo número de elementos que o fluxo original.
`collect`	Cria uma *nova coleção* dos elementos contendo os resultados das operações anteriores do fluxo.
`filter`	Resulta em um fluxo contendo apenas os elementos que atendem uma condição.
`String::toUpperCase`	Referência de método para um método de instância de uma classe. Cria uma lambda de um parâmetro que chama o método de instância sobre o argumento da lambda e retorna o resultado do método. Usada na Figura 17.7.

Referência Bibliográfica Principal

• DEVMEDIA. Disponível em https://www.devmedia.com.br. Acessado em Julho de 2019.

• DEITEL, Harvey M. Java: Como Programar – 10 ed. Cap 17. 2015.

• CAELUM. Disponível em https://www.caelum.com.br/apostila-java-orientacao-objetos/classes-abstratas/. Acessado em Agosto de 2019.