A B A K Ó S
PUC Minas Instituto de Ciências Exatas e Informática

Lambda Expressions and functional interfaces *

Ravi Antônio Gonçalves de Assis[1]
Fernando Pimenta de Figueiredo Pontello[2]
Jônathas Leandro Sousa[3]
Mateus Ferreira da silva[4]

1 INTRODUÇÃO

Em meados de 2015, com o surgimento do Java 8, a linguagem Java recebeu importantes e significativas atualizações, com inclusão de diversos novos recursos e funcionalidades. Destes recursos, será abordado neste documento a Lambda Expression, em português, expressões lambdas. As expressões lambdas são um recurso comum em linguagens funcionais com JavaScript, Haskell, Lisp entre outras, onde o foco seria, basicamente, tratar a programação como funções matemáticas. Além de adicionar novos elementos de sintaxe, o incremento deste novo recurso otimiza a implementação de determinadas estruturas.

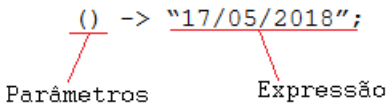
* Artigo apresentado à Revista Abakos
[1] Graduação em Engenharia de Software da PUC Minas, Brasil.
[2], Graduação em Engenharia de Software da PUC Minas, Brasil.
[3],Graduação em Engenharia de Software da PUC Minas, Brasil.
[4],Graduação em Engenharia de Software da PUC Minas, Brasil.

2 O QUE É UMA EXPRESSÃO LAMBDA E INTERFACES FUNCIONAIS?

Expressão lambda é, basicamente, um método anônimo que implementa uma interface funcional. Se assemelham muito com as antigas classes anônimas, na tarefa de implementar interfaces funcionais, mas são mais concisas e mais fáceis de serem utilizadas. Um exemplo é que expressões lambdas implementam interfaces funcionais que para serem utilizadas posteriormente. Interfaces funcionais são interfaces que contém apenas um método abstrato e que especifica a finalidade pretendida para a interface. Elas costumam representar uma única ação. Exemplo: Interface Runnable é uma interface funcional porque só define um método: run(),ultilizado como base o livro (BLOCH, 2018) como base de pesquisa.

3 SINTAXE EXPRESSÕES LAMBDA.

As expressões lambda introduzem o elemento de sintaxe '->', conhecido como operador lambda ou operador seta. Ele divide a expressão lambda da seguinte forma: do lado esquerdo ao operador lambda (->) é especificado os parâmetros da expressão. Do lado direito é especificado as ações da expressão. Exemplo básico de uma expressão lambda: A expressão acima se assemelha ao método abaixo:



String estreiaDeadpool2() { **return** "17/05/2018";}

Quando a expressão lambda requer um parâmetro, ele é definido no lado esquerdo do operador ' -> '

(num) -> num%2 == 0;

A expressão acima também poderia ser escrita das seguintes formas:

- $num->num\%2 == 0$; //Não precisa de parênteses para um argumento apenas
- $(num)->$ { **return** $num\%2 == 0$; //A expressão pode vir dentro de um bloco de código }

Nota: Caso seja utilizado o bloco de código, é necessário utilizar o **return**.

4 IMPLEMENTANDO A INTERFACE FUNCIONAL

Como mencionado anteriormente, uma interface funcional é aquela que especifica apenas um método abstrato. A interface pode incluir métodos padrão e/ou static, mas somente um método abstrato. Exemplo de interface funcional:

$$interface\ DeadPool2\{$$
$$String\ estreia();\}$$

Nota: Não é necessário utilizar o modificador $abstract$ num método de uma interface. Ele já é definido implicitamente como abstrato.

Atribuindo a expressão lambda à referência da interface funcional:

$$DeadPool2\ film = () -> \text{"17/05/2018"};$$

Atribuição acima é correta pois a expressão lambda é compatível com o método estreia(). Ambas não possuem parâmetros e retornam uma String. As lambdas para serem atribuídas a interfaces funcionais devem ser compatíveis com o método da interface funcional. O exemplo abaixo não será compilado por não atender a definição do método da interface:

//Não vai compilar

$$DeadPool2\ film = (num) -> num;$$

5 UTILIZANDO AS EXPRESSÕES LAMBDAS

Vamos apresentar algumas formas de utilização das expressões lambdas. Considere a interface funcional a seguir:

$$interface\ TestNum\{\ \textbf{boolean}\ test(int\ n, int\ m);\ \}$$

Um ponto importante da interface funcional é que ela pode ser usada com qualquer expressão lambda com a qual seja compatível. Para demonstrar isso, apresentaremos um código que utilizará a interface funcional TestNum para realizar três testes diferentes. Para cada teste, será implementado uma expressão lambda que seja compatível com o método test(int, int) da interface funcional.

public static void $main\ (String\ args[])\{$

$TestNum\ igual = (m, n) -> m == n;$

$if(igual.test(5, 5))System.\textbf{out}.println(\text{“Numeros iguais”});$

$if(!igual.test(10,2))System.\textbf{out}.println(\text{“Numeros diferentes”});$

$TestNum\ menor = (m,n) -> m < n;$

$if(menor.test(10,5))System.\textbf{out}.println(\text{“10 é menor que 5”});$

$if(!menor.test(5,10))System.\textbf{out}.println(\text{“5 não é menor que 10”});$

$TestNum\ fator = (n,d) -> n\%d == 0;$

$if(fator.test(10,2))System.\textbf{out}.println(\text{“10 é múltiplo de 2”});$

$if(!fator.test(10,3))System.\textbf{out}.println(\text{“10 não é múltiplo de 3”});$ }

6 INTERFACE FUNCIONAL GENÉRICA

As interfaces funcionais podem receber parâmetros de tipo. Dessa forma, podem implementar uma interface funcional genérica. Veja o exemplo a seguir:

$interface\ Test<T>\{$

boolean $igual(Tm,Tn);$ }

class Demo2{

public static void main(String args[]){

$Test<Integer>\ testInt = (m,n) -> m == n;$

$if(testInt.igual(5,5))\ System.\textbf{out}.println(\text{“Numeros iguais”});$

$Test<String>\ testStr = (s1,s2) -> s1.compareTo(s2) == 0;$

$if(testStr.igual(\text{“abc”, “abc”})\ )\ System.\textbf{out}.println(\text{“Frases iguais”});$

} }

Como mostra o exemplo acima, ao utilizar interfaces funcionais genéricas, podemos definir uma única interface funcional capaz de comparar se dois termos são iguais, independente do tipo dos termos.

7 REFERÊNCIA DE MÉTODOS

Uma referência de método é uma forma de referenciar um método sem executá-lo. Como as expressões lambdas, a referência de método precisa de uma interface funcional compatível. Para a referência de métodos é definido um novo operador, o operador ‘ :: ’. Exemplo de sintaxe básica:

$$NomeClasse :: nomeMetodo$$

O exemplo acima é de uma referência de um método estático. As referências de métodos também podem ser de métodos de instâncias ou de um construtor de uma classe.

$ObjInstancia : metodoInstancia$ //referencia metodo instancia.
$NomeClasse :: new$ //referencia ao construtor de uma classe.

O exemplo abaixo faz uma referência de um método estático.

```
//interface Funcional
interface NumInt {
    boolean test(intn); }

class TestInt {
    //testa se n é positivo
static boolean positivo(intn){ return n > 0; }
    // testa se n é par
static boolean par(int n){ return n } }

class Demo3 {
    //método que possui uma interface funcional como parametro
    //logo pode receber uma expressão lambda ou uma referência
    //de metodo compatível com a interface
static boolean numTest(NumIntp, intv){
    return p.test(v); }

public static void main(String args[]){
    int num = 10;
    //testa se 5 é positivo. Passa a referência do método
    //positivo como parâmetro
    if(numTest(TestInt :: positivo, num))
        System.out.println(num+“ é positivo”);
    else
        System.out.println(num +“ é negativo”);
    if(numTest(TestInt :: par, num))
```

```
        System.out.println(num+ " é par");
    else
        System.out.println(num+ " é impar" } }
```

8 STREAMS API

Uma Stream representa uma sequência de elementos. Ela dá suporte a várias operações em sequência nesses elementos. Essas sequências de operações é chamada de pipeline. Segue um exemplo:

$List < Integer > numeros = Arrays.asList(1, 2, 3, 4, 5);$
$numeros.stream()$
$.filter(n- > n > 2)$
$.forEach(System.\mathbf{out}::println);$

O exemplo anterior poderia ser escrito da seguinte maneira sem o uso de streams:

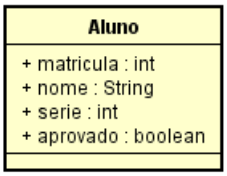
for$(Integern : numeros)\{$
$if(n > 2)$ { *System.**out**.println(n)*;
}

Como podem ver, o exemplo utilizando streams apresenta um código mais conciso, genérico e legível de manipular coleções de dados.

9 MANIPULANDO UMA COLLECTION UTILIZANDO STREAMS

Para ilustrar o uso de streams para manipular coleções, vamos utilizar a seguinte classe Aluno.

Figura 1 – Classe Aluno



Utilizando a classe Aluno, podemos criar uma coleção do tipo Aluno e utilizar de Streams para manipular essa coleção.

$List < Aluno > classe$ = ArrayList<Aluno>();
$classe.add(new\ Aluno$(1,"Antônio", 8, **true**));
$classe.add(new\ Aluno$(2,"Geraldo", 8, **false**));
$classe.add(new\ Aluno$(3,"Julia", 8, **true**));
$classe.add(new\ Aluno$(4,"Eliza", 8, **true**));

Filtrando os alunos aprovados:

$classe.stream()$

$.filter(aluno- > aluno.aprovado)$

$.forEach(aluno- > System.\mathbf{out}.println(aluno))$;

Imprimindo os alunos em ordem alfabética:

$classe.stream()$

$.sorted((x, y)- > x.nome.compareTo(y.nome))$

$.forEach(aluno- > System.\mathbf{out}.println(aluno))$

Pesquisando apenas um aluno

$classe.stream()$

$.filter((3, aluno)- > aluno.matricula == 3)$

$.forEach(aluno- > System.\mathbf{out}.println(aluno))$;

Referências

BLOCH, Joshua. **Effective Java**. 3. ed. [S.l.]: Pearson Education Inc., 2018.