SENAI

Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

Funções Lambda (java)

Prof.: Giovanni Leopoldo Rozza

SENAI
PELO FUTURO DO TRABALHO

Funções Lambda

As funções lambda foram introduzidas na versão 8 do Java, lançada em março de 2014. Elas foram adicionadas para facilitar a programação funcional e melhorar a legibilidade do código.

A programação funcional é um **paradigma que visa estruturar a construção de softwares seguindo o modelo de funções matemáticas**, objetivando a imutabilidade (não alteram o estado dos objetos) dos dados e o acoplamento de funções, que resultam em benefícios como consistência tanto do código quanto dos dados.

Semelhante a uma função matemática, u**m código funcional tem a sua saída condicionada exclusivamente à sua entrada, de modo que uma entrada garanta sempre a mesma saída após a execução do mesmo método**, eliminando os chamados side-effects, que são as mudanças de estado ocasionadas por outros fatores que não dependam dos parâmetros do método.

Funções Lambda

O paradigma imperativo versus o paradigma funcional

Paradigma imperativo

Sequência de instruções (comandos) que alteram variáveis em memória.

```
int total = 0;
for (int i=0; i< 10; i++){
    total += i;
}
```

Paradigma funcional

- Conjunto de definições de funções que aplicamos a valores;
- Promove imutabilidade.

```
double volCilindro(double r, double h)
{
    return areaCirculo(r)*h;
}


double areaCirculo(double r){
    return Math.PI*r*r;
}
```

Funções Lambda

Programação em Java 7 era possível, mas muito verboso.

```
List<Familia> familias = Arrays.asList(
  new Familia("Stark", "Winterfel"),
  new Familia("Lannister", "Lanisporto"),
  new Familia("Baratheon", "Ponta Tempestade"),
  new Familia("Targaryen", "Pedra do Dragão"));

Collections.sort(familias, new Comparator<Familia>() {
  @Override
  public int compare(Familia f1, Familia f2) {
    return f1.getNome().compareTo(f2.getNome());
  }
});
```

classe anônima

Interface *Comparator*
Tem um único método "compare"

Funções Lambda

Uma expressão lambda é um pequeno bloco de código que recebe parâmetros e retorna um valor. As expressões lambda são semelhantes aos métodos, mas não precisam de nome e podem ser implementadas diretamente no corpo de um método.

SINTAXE:

A expressão lambda mais simples contém um único parâmetro e uma expressão:

```
parameter -> expression
```

Para usar mais de um parâmetro, coloque-os entre parênteses:

```
(parameter1, parameter2) -> expression
```

Funções Lambda

As expressões são limitadas. Eles devem retornar imediatamente um valor e não podem conter variáveis, atribuições ou instruções como `if` ou `for`.

Para realizar operações mais complexas, um bloco de código pode ser usado com chaves. Se a expressão lambda precisar retornar um valor, o bloco de código deverá ter uma instrução de `return`

```
(parameter1, parameter2) -> { code block }
```

Funções Lambda

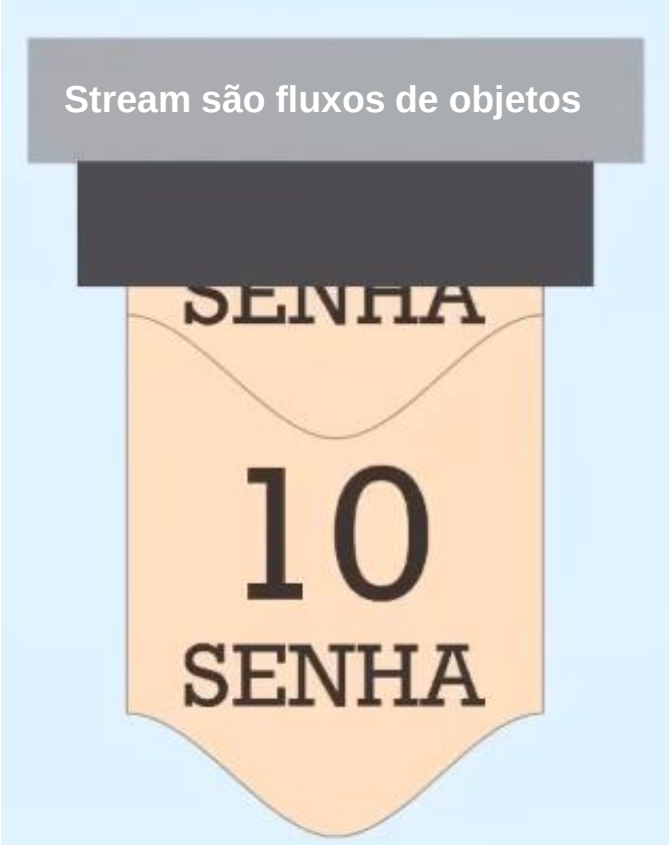
Usando agora a função Lambda para ordenar uma *Collection*, fica menos verboso:

```
List<Familia> familias = Arrays.asList(
  new Familia("Stark", "Winterfel"),
  new Familia("Lannister", "Lanisporto"),
  new Familia("Baratheon", "Ponta Tempestade"),
  new Familia("Targaryen", "Pedra do Dragão"));

Collections.sort(familias, (f1, f2) -> f1.getNome().compareTo(f2.getNome()));
```

Funções Lambda

As funções Lamba facilitaram muito o uso da programação funcional no Java 8, mas esse "poder" tem um origem: Streams



Funções Lambda

EXEMPLO 1: BUSCAR UMA PALAVRA EM UM ARQUIVO TEXTO

```java
public class Main {

    /**
     * BUSCA ARQUIVO (OLD WAY)
     */
    no usages
    public static void main(String[] args) throws Exception {

        String palavra = "Antonio";
        String arquivo = "nomes.txt";

        try (BufferedReader bufferedReader = new BufferedReader(new FileReader(arquivo))) {
            long contador = 0;      // variável em memória
            String linha = null;    // variável em memória

            while ((linha = bufferedReader.readLine()) != null) {
                if (linha.contains(palavra)) {
                    contador++;
                }
            }

            System.out.printf("A palavra [%s] ocorreu %d vezes\n", palavra, contador);
        }
    }
```

Funções Lambda

EXEMPLO 1: BUSCAR UMA PALAVRA EM UM ARQUIVO TEXTO

```java
public static void main(String[] args) throws Exception {

    String palavra = "Antonio";
    String arquivo = "nomes.txt";
    long contador = 0;

    // lê todo o arquivo de uma vez!
    List<String> linhas = Files.readAllLines(Paths.get(arquivo), StandardCharsets.UTF_8);

    for( String linha:linhas) {
        if( linha.contains(palavra)) {
            contador++;
        }
    }
        System.out.printf("A palavra [%s] ocorreu %d vezes\n", palavra, contador);
    }
}
```

Funções Lambda

EXEMPLO 1: BUSCAR UMA PALAVRA EM UM ARQUIVO TEXTO

```
public static void main(String[] args)
{
    String palavra = "Antonio";
    String arquivo = "nomes.txt";

    //  Usando stream e funcao lamba
    // Não necessariamente carrega, filtra e conta,
    // decide a melhor hora de fazer inclusive
    // pode fazer EM PARALELO
    // try-with-resources ira fechar automaticamente o Stream<String>
    // retornado pelo Files.lines().
    // O Stream é aberto dentro do bloco try e
    // será fechado automaticamente quando o bloco for encerrado.
    try (Stream<String> lines = Files.lines(Paths.get(arquivo), StandardCharsets.UTF_8))
    {
        long contador = lines.filter(linha -> linha.contains(palavra)).count();
        System.out.printf("A palavra [%s] ocorreu %d vezes\n", palavra, contador);
    } catch (IOException e) {
        System.err.println("Ocorreu um problema ao ler o arquivo: " + e.getMessage());
    }

}
}
```

Funções Lambda

EXEMPLO 1: O QUE MELHOROU A PROGRAMAÇÃO FUNCIONAL?

- Sem variáveis de controle explícitas
- Eficiência e melhor legibilidade
- Usando streams e filtros

Funções Lambda

EXEMPLO 2: BUSCAR CLIENTES EM UMA LISTA AGRUPADO POR ESTADO

- Sintaxe mais concisa
- Altamente expressiva
- Uso de funções especializadas

Funções Lambda

EXEMPLO 3: COMPOSICÃO DE REGRAS

- Estrutura orientada a objetos complexa para funcional simplificada
- Usando composição de operações

Funções Lambda

A interface funcional **Predicate** é parte do pacote **java.util.function** e é usada para representar uma função que recebe um argumento e retorna um valor booleano. Essa interface é frequentemente usada para testar condições em coleções de objetos.
A interface Predicate **possui um único método chamado "test",** que recebe um argumento e retorna true ou false com base em uma condição específica. Assim por exemplo pode-se utilizar um Predicate para verificar se um número é par ou ímpar, ou se uma string contém um determinado padrão.
Uma das principais vantagens de usar a interface Predicate é a capacidade de combinar várias condições usando os métodos padrão fornecidos pela interface. Alguns desses métodos incluem "and", "or" e "negate", que permitem criar expressões lógicas complexas.
Além disso, a interface Predicate também pode ser usada em conjunto com outras classes e interfaces do Java, como List, Set e Map, para filtrar elementos com base em determinadas condições.

Uma interface funcional em Java é uma interface que **possui apenas um único método abstrato**. Essa característica permite que **ela possa ser usada como um tipo de dado para expressões lambda ou referências a métodos**.
Em outras palavras, uma interface funcional define um **contrato** para uma função que pode ser **passada como argumento, retornada por um método ou atribuída a uma variável**. Ela fornece uma maneira concisa e flexível de representar comportamentos específicos a partir do Java 8.

Funções Lambda

Exemplo de algumas interfaces funcionais em Java:

```
Runnable runnable = () -> {
      // código a ser executado
  };

  Comparator<String> comparator = (str1, str2) -> str1.compareTo(str2);

  Predicate<Integer> numeroPar = num -> num % 2 == 0;

  Function<Integer, String> integerToString = num -> String.valueOf(num);

  Consumer<String> printMensagem = messagem -> System.out.println(messagem);

  Supplier<Double> obtemNumeroRandomico = () -> Math.random();

  UnaryOperator<Integer> AreaAquadrado = num -> num * num;
```

Funções Lambda

EXEMPLO 4: LAZY LOADING ARQUIVO CSV EM MEMÓRIA

- Filtro e conversão de dados
- Execução de forma lazy

SENAI

Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

0800 048 1212 **sc.senai.br**

Rodovia Admar Gonzaga, 2765 - Itacorubi - 88034-001 - Florianópolis, SC