



# Módulo 20





# BackEnd Java



Rodrigo Pires





# O que iremos ver:

- Métodos Default.
- Interfaces Funcionais
- Expressões Lambdas.
- Parênteses e colchetes com expressões lambda, quando usar e mais.
- Referências de método para lambda.





# Expressões Lambdas

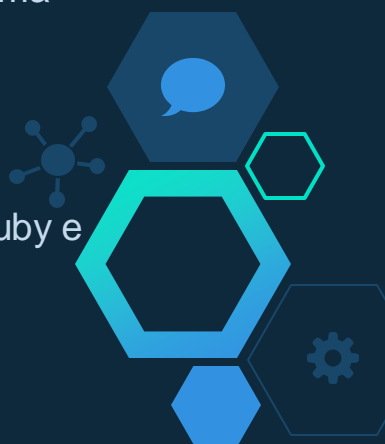


# O que são?

Expressões Lambda são funções anônimas.

A principal diferença de uma função normal e uma função lambda é que elas não possuem nome, sendo definidas diretamente em uma única linha, fornecendo abstrações para problemas complexos limpando o código deixando o escopo menor.

Elas estão presentes nas linguagens como Java, C#, Python, Ruby e entre outras do mercado.





# Vantagens e desvantagens

- Vantagens:
  - ◆ Código mais simples.
  - ◆ Simplifica diversas operações com coleções de dados.
- Desvantagens:
  - ◆ Funções anônimas podem gerar problemas na hora de depurar o código.
  - ◆ Muitos argumentos em uma Expressão Lambda gera legibilidade mais difícil.





# Identificando expressões lambdas

Quando é encontrado o símbolo ->

```
palavras.forEach((String s) -> {  
    System.out.println("Consumer " + s);  
})  
);
```



A decorative graphic on the left side of the slide. It features a large central hexagon with a white number '2'. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines.

2

# Interfaces Funcionais

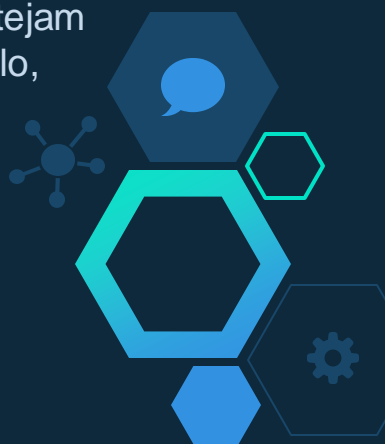




# O que são?

Interfaces funcionais são interfaces que têm um método a ser implementado, um método abstrato. Isso significa que toda interface criada que respeite esta premissa, torna-se automaticamente uma interface funcional.

O compilador reconhece essas interfaces e permite que elas estejam disponíveis para que os desenvolvedores trabalhem, por exemplo, com expressões lambda.





# Interfaces novas a partir do Java 8

- Fornecedor / Supplier
- Consumidor e BiConsumidor / Consumer e BiConsumer
- Predicado e BiPredicado / Predicate e BiPredicate
- Função e BiFunção / Function e BiFunction
- UnaryOperator e BinaryOperator

Todas elas se encontram no pacote **java.util.function**



A decorative graphic on the left side of the slide. It features a large central hexagon with a blue-to-teal gradient, containing the white number '3'. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and teal. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines, and a speech bubble icon.

3

# Expressões Lambdas



# O que são?

```
(int a, int b) -> { return a + b; }
```

```
() -> System.out.println("Hello World");
```

```
(String s) -> { System.out.println(s); }
```

```
() -> 42 () -> { return 3.1415; }
```

```
a -> a > 10
```





# Exemplo com Thread

```
public static void main(String[] args) {  
    Runnable r = new Runnable() {  
        public void run() {  
            System.out.println("Thread com classer interna!");  
        };  
    };  
    new Thread(r).start();  
}
```





# Exemplo com Thread

```
Runnable r1 = () -> System.out.println("Thread com função lambda!");  
new Thread(r1).start();|
```





# Exemplo com Thread

```
new Thread( () -> System.out.println("hello world") ).start();
```





# Lambda X Interfaces anônimas

```
@FunctionalInterface
public interface MyEventConsumer {

    public void consume(Object event);
}
```

```
MyEventConsumer myCon = new MyEventConsumer() {
    @Override
    public void consume(Object event) {
        System.out.println(event);
    }
};

myCon.consume(event: "Olá Rodrigo");
```







# Lambda X Interfaces anônimas

```
MyEventConsumer myConLam = (value) -> System.out.println(value);  
myConLam.consume(event: "Olá Rodrigo com Lambda");
```






# Lambda X Interfaces anônimas

```
//  
recebeInterface((x) -> System.out.println(x));
```

```
private static void recebeInterface(MyEventConsumer consumer) {  
    consumer.consume(event: "Olá novamente");  
}
```



A decorative graphic on the left side of the slide. It features a large central hexagon with a blue-to-cyan gradient, containing the white number '4'. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines, and a speech bubble icon.

4

## Parênteses e colchetes



# Parênteses e colchetes

```
@FunctionalInterface  
public interface MyEventConsumer {  
  
    public void consume(Object event);  
}
```

```
MyEventConsumer myConLamBody = s -> {  
    System.out.println(s);  
};  
myConLamBody.consume( event: "Com body");
```

```
MyEventConsumer myConLamBody = (s) -> {  
    System.out.println(s);  
};  
myConLamBody.consume( event: "Com body");
```



# Retornando valores

```
@FunctionalInterface
```

```
public interface MyEventConsumer1 {
```

```
    public Integer consume(Integer value, Integer value1);
```

```
}
```

```
MyEventConsumer1 myConLamBody = (value1, value2) -> {
```

```
    System.out.println(value1);
```

```
    System.out.println(value2);
```

```
    return value1 + value2;
```

```
};
```

```
Integer result = myConLamBody.consume(value: 10, value1: 20);
```

```
System.out.println("Result " + result);
```



A decorative graphic on the left side of the slide, consisting of a cluster of hexagons in various shades of blue and cyan. Some hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. A network of dots and lines is also visible.

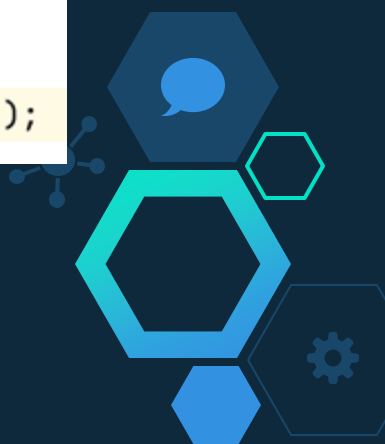
4

## Referências de método para lambda



# Referência de método

```
//Metodo reference  
MyEventConsumer myConLamRe = System.out::println;  
myConLamRe.consume( event: "Olá Rodrigo com Referência de método");
```






# Referência de método

```
//Referência de método estático
Finder finder = MyClass::doFind;
int result = finder.find("Teste", "Teste");
System.out.println(result);

//Referência de método no parâmetro
Finder finderParam = String::indexOf;
int resultParam = finderParam.find("Teste", "Teste");
System.out.println(resultParam);
//Mesma coisa que a linha de cima, só que com lambda
Finder finderParamSame = (s1, s2) -> s1.indexOf(s2);
int resultParamLamb = finderParamSame.find("Teste", "Teste");
System.out.println(resultParamLamb);

//Referência de método em construtores
Factory factory = String::new;
String ret = factory.create("Olá");
System.out.println(ret);
//Mesma coisa que a linha de cima, só que com lambda
Factory factory1 = s -> new String(s);
String ret1 = factory1.create("Olá");
System.out.println(ret1);
```







# Referências

[Exemplos disponíveis no meu github:](#)

<https://github.com/digaomilleniun/backend-java-ebac>

