# Java Avançado

Expressões Lambda


## Tópicos Abordados


- O que são expressões lambda
- Exemplos de uso
- Anatomia
- Sintaxe
- Interfaces funcionais
  - *Predicate<T>*
  - *Consumer<T>*
  - *Function<T, R>*
- Novas funcionalidades em coleções
- Referências a métodos
- Closures

## Expressões Lambda


- Maior inovação da versão 8 do Java
- O nome lambda vem do conceito matemático de *cálculo lambda*
- Expressões lambda trazem o Java mais próximo do paradigma de programação funcional
- Uma expressão lambda pode ser considerada como um objeto
  - Pode ser referenciada por uma variável
  - Pode ser passada como parâmetro para métodos e utilizada como retorno
- Em Java, uma expressão lambda é utilizada em substituição a uma inner class anônima

## Exemplo de utilização

```
Runnable r = new Runnable() {
  @Override
  public void run() {
    System.out.println("ABC");
  }
};

new Thread(r).start();
```

Inner class anônima

```
Runnable r = () -> System.out.println("ABC");
new Thread(r).start();
```

Expressão lambda

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

Expressão lambda

Resultado: código mais simples e intuitivo

## Anatomia de uma expressão lambda

- Uma expressão lambda é representada da seguinte forma

```
(Parâmetros) -> { Corpo }
```

Operador *arrow*

## Exemplos de sintaxe

```
public interface Testable {
  public boolean test(int x);
}
```

```
Testable t =
```

```
e -> e > 10
```

```
(e) -> (e > 10)
```

```
(int e) -> e > 10;
```

```
e -> {
  return e > 10;
}
```

```
public interface Calculator {
  public int calculate(int a, int b);
}
```

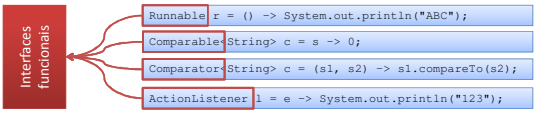
```
Calculator c =
```

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

```
(x, y) -> {
  x = x + 1;
  y = y - 1;
  return x + y;
}
```

## Interfaces funcionais

- Uma interface funcional tem duas características
  1. É uma interface
  2. Possui apenas 1 método
- Uma expressão lambda pode ser atribuída a uma variável de uma interface funcional




Interfaces que já pertenciam ao Java agora passam a ser interfaces funcionais

## *@FunctionalInterface*

- Esta annotation define uma interface como interface funcional

```
@FunctionalInterface
public interface Generator {
  public String generate();
}
```

- Seu uso não é obrigatório
- Se *@FunctionalInterface* for utilizada, o compilador checa se a interface define apenas 1 método

## Interface funcional: *Predicate<T>*

```
@FunctionalInterface
public interface Predicate<T> {
  boolean test(T t);
}
```

- O tipo *T* define o tipo a ser testado
- Retorna um *boolean*

```
Predicate<String> p = s -> s.length() > 5;
```

*true* se o tamanho da *String* for maior do que 5

## Interface funcional: *Consumer<T>*

```
@FunctionalInterface
public interface Consumer<T> {
  void accept(T t);
}
```

- O tipo *T* define o tipo a ser processado
- Não retorna informação (*void*)

```
Consumer<Integer> c = i -> System.out.println(i);
```

Processa *i* escrevendo o seu valor no console

## Interface funcional: *Function<T, R>*

```
@FunctionalInterface
public interface Function<T, R> {
  R apply(T t);
}
```

- O tipo *T* define o tipo de origem
- O tipo R define o tipo de destino a ser retornado

```
Function<String, Integer> f = s -> Integer.parseInt(s);
```

Transforma uma *String* em um *int*

## Novas funcionalidades em coleções

- A Collections API ganhou novas funcionalidades para aproveitar o uso de expressões lambda

```
List<Integer> l = new ArrayList<>();
l.add(1);
l.add(2);
l.add(3);
```

```
l.forEach(item -> System.out.println(item));
```

*Consumer<T>*

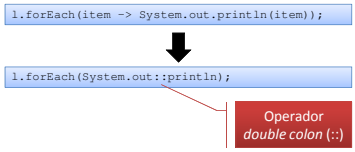
```
l.removeIf(item -> item % 2 == 0);
```

*Predicate<T>*

## Referências a métodos

- Permite converter métodos já existentes em expressões lambda

```
l.forEach(item -> System.out.println(item));
```



```
l.forEach(System.out::println);
```

Operador *double colon* (::)

Os parâmetros da expressão lambda são repassados para o método

## Closures

- Expressões lambda têm a capacidade de acessar variáveis definidas externamente
- Este recurso é denominado *closure*

```
int mult = 2;
Function<Integer, Integer> f = (x -> x * mult);

System.out.println(f.apply(5));
```

Variável definida fora da expressão lambda

Imprime o valor 10

Variáveis externas acessadas por expressões lambda são implicitamente definidas como *final*

## Closures e atributos

- No caso da variável externa ser um atributo de classe, o *final* implícito não se aplica
- O valor considerado é o do momento da execução

```
public class MyClass {
  private int mult = 2;

  public void executar() {
    Function<Integer, Integer> f = (x -> x * mult);
    mult = 5;
    System.out.println(f.apply(5));
  }
}
```

Referencia o atributo

Imprime o valor 25


Softblue