Prof. M.Sc. Reinaldo Jr

#LAMBDA

LAMBDA

- É uma maneira mais simples de implementar uma interface que só possui um método.

LAMBDA [ANTES]

```
1 class Mostrador implements Consumer<Usuario> {
2     public void accept(Usuario u) {
3         System.out.println(u.getNome());
4     }
5 }
```

```
1 Mostrador mostrador = new Mostrador();
2 usuarios.forEach(mostrador);
```

LAMBDA [DEPOIS I]

```
1 Consumer<Usuario> mostrador =
2         u -> System.out.println(u.getNome());
```

```
usuarios.forEach(mostrador);
```

LAMBDA [DEPOIS II]

```
usuarios.forEach(u -> System.out.println(u.getNome()));
```

- Podemos dizer então que toda interface do Java que possui apenas um método abstrato pode ser instanciada como um código lambda!

LAMBDA

- Podemos dizer então que toda interface do Java que **possui apenas um método abstrato** podemos chamar de **INTERFACE FUNCIONAL**.

#INTERFACE FUNCIONAL

INTERFACE FUNCIONAL

```
1 @FunctionalInterface
2 interface Validador<T> {
3     boolean valida(T t);
4 }
```

INTERFACE FUNCIONAL

```
1 @FunctionalInterface
2 interface Validador<T> {
3     boolean valida(T t);
4     boolean outroMetodo(T t);
5 }
```

Ao compilar esse código, recebemos o seguinte erro:

```
1 java: Unexpected @FunctionalInterface annota-
tion
2   Validador is not a functional interface
3   multiple non-overriding abstract methods
found in interface Exemplo
```

INTERFACE FUNCIONAL

```
1 @FunctionalInterface
2 interface Validador<T> {
3     boolean valida(T t);
4 }
```

- A anotação serve apenas para que ninguem torne aquela interface não-funcional acidentalmente.

#DEFAULT METHOD

DEFAULT METHOD

- Tomamos o cuidado, pois uma interface funcional é aquela que possui apenas um método abstrato! **Ela pode ter sim mais métodos, desde que sejam métodos default**.
- **Métodos com código, dentro de interfaces** ☺

DEFAULT METHOD

```
1 @FunctionalInterface
2 public interface Consumer<T> {
3       void accept(T t);
4
5       default Consumer<T> andThen(Consumer<? super
T> after) {
6             Objects.requireNonNull(after);
7             return (T t) -> { accept(t); after.accept(t); };
8       }
9 }
```

Repare que por ser um método default, a classe pode ser anotada com @FunctionalInterface

DEFAULT METHOD

```
// Utilizando o método default "andThen"
listaClientes.forEach(
                        ( (Consumer<Cliente>) c->System.out.println("Antes de Imprimir") ).
                           andThen(c -> System.out.println(c.getNome() + "-" + c.getCpf()))
                      );
```

#ORDENAÇÃO COM LAMBDA

ORDENAÇÃO COM LAMBDA

```
// ORDENAÇÃO SEM LAMBDA
Comparator<Cliente> comparador = new Comparator<Cliente>() {
    public int compare(Cliente c1, Cliente c2) {
    return c1.getNome().compareTo(c2.getNome());
}};

listaClientes.forEach((c->System.out.println(c.getNome())));
Collections.sort(listaClientes, comparador);
listaClientes.forEach((c->System.out.println(c.getNome())));


// ORDENAÇÃO COM LAMBDA
listaClientes.forEach((c->System.out.println(c.getNome())));
Collections.sort(listaClientes, (c1, c2) -> c1.getNome().compareTo(c2.getNome()));
listaClientes.forEach((c->System.out.println(c.getNome())));
```