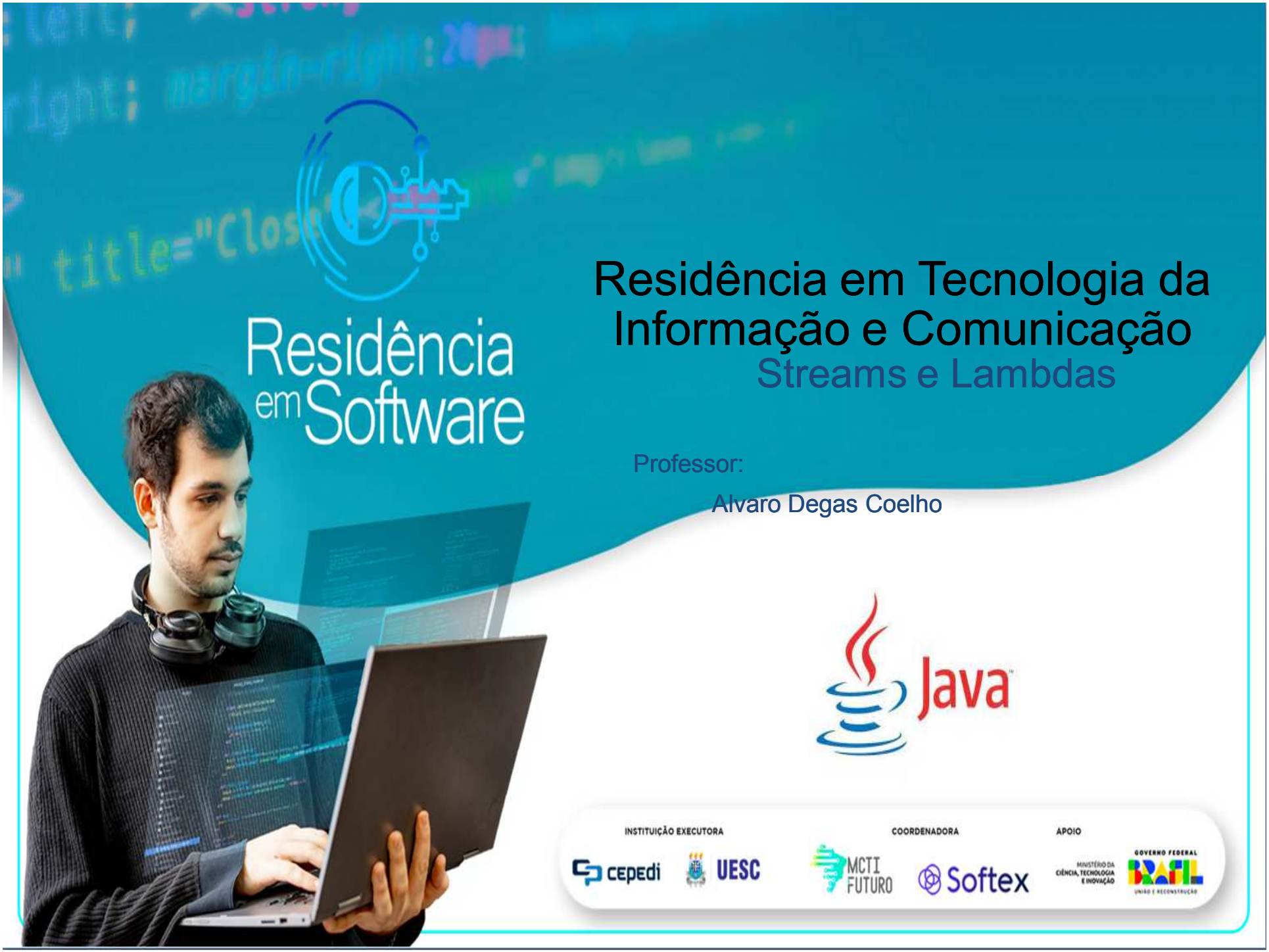
Residência em Software
Residência em Tecnologia da Informação e Comunicação
Streams e Lambdas
Professor:
Alvaro Degas Coelho
Java
INSTITUIÇÃO EXECUTORA
cepedi
UESC
COORDENADORA
MCTI FUTURO
Softex
APOIO
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Um programa em... HASKELL

- Observe o código abaixo
  - do { n <- readLn ; print (n^2) }
- Lê um valor (guarda em n) e escreve este valor ao quadrado

Mais um tiquinho de HASKELL

- Este trecho aqui: o que faz?
  - factorial n = if n == 0 then 1 else n * factorial (n - 1)
- E este código?

```
main = do putStrLn "Quanto é 5! ?“
x <- readLn
if x == factorial 5
then putStrLn "Você acertou!“
else putStrLn "Você errou!"
```

Residência em Software

Cálculo Lambda

- Alonzo Church
- Trabalho paralelo ao de Alan Turing
- Sistema formal para expressar computação baseado em abstração de funções e aplicação usando apenas atribuições de nomes e substituições .
- Linguagens funcionais
  - LISP e HASKELL (mais Miranda, Elixir, Scheme...)

Expressões Lambda em Java

- Um lambda é uma lista de parâmetros seguida por uma seta e deopis um corpo de programa
  - (parameterList) -> {statements}
- Exemplo
  - (int x, int y) -> {return x + y;}
- Retorna a soma de dois números

Residência em Software

Expressões Lambda

- O corpo do Lambda pode conter várias instruções cercadas por chaves
- O tipo de dados do parâmetro pode ser omitido
  - (x, y) -> {return x + y;}
- O que força o retorno a ser determinado pelo contexto

Residência em Software

Expressões Lambda

- Um lambda coom apenas uma expressão como corpo pode ser escrita diretamente
  - (x, y) -> x + y
- O valor da expressão é retornado implicitamente

Residência em Software

Expressões Lambda

- Quando a lista de parâmetros é unitária, pode-se omitir os parênteses
  - value -> System.out.printf("%d ", value)
- Um lambda com uma lista de parâmetros vazia é definida como ()
  - () -> System.out.println("Welcome to lambdas!")

Streams

- Streams são objetos que implementam a interface Stream
  - package: java.util.stream
- Permite executar instruçõs de programação funcional
- Há interfaces especializadas para int, long e double

Streams

- Streams movem elementos através de uma sequência de passos
  - É o stream pipeline
- O pipeline começa com uma fonte de dados
- Executa operações intermediárias
- Encerra com uma operação terminal

Operações Terminais

- Inicia o processamento das operações intermediárias
- Produz um resultado
- Eles requisitam as operações quando são invocados

Operações Intermediárias

Intermediate `Stream` operations	
`filter`	Results in a stream containing only the elements that satisfy a condition.
`distinct`	Results in a stream containing only the unique elements.
`limit`	Results in a stream with the specified number of elements from the beginning of the original stream.
`map`	Results in a stream in which each element of the original stream is mapped to a new value (possibly of a different type)—e.g., mapping numeric values to the squares of the numeric values. The new stream has the same number of elements as the original stream.
`sorted`	Results in a stream in which the elements are in sorted order. The new stream has the same number of elements as the original stream.

Operações Terminais

Terminal `Stream` operations	
`forEach`	Performs processing on every element in a stream (e.g., display each element).
Reduction operations—*Take all values in the stream and return a single value*	
`average`	Calculates the *average* of the elements in a numeric stream.
`count`	Returns the *number of elements* in the stream.
`max`	Locates the *largest* value in a numeric stream.
`min`	Locates the *smallest* value in a numeric stream.
`reduce`	Reduces the elements of a collection to a *single value* using an associative accumulation function (e.g., a lambda that adds two elements).
Mutable reduction operations—*Create a container (such as a collection or* `StringBuilder`*)*	
`collect`	Creates a *new collection* of elements containing the results of the stream's prior operations.
`toArray`	Creates an *array* containing the results of the stream's prior operations.

Operações Terminais

Terminal `Stream` operations	
Search operations	
`findFirst`	Finds the *first* stream element based on the prior intermediate operations; immediately terminates processing of the stream pipeline once such an element is found.
`findAny`	Finds *any* stream element based on the prior intermediate operations; immediately terminates processing of the stream pipeline once such an element is found.
`anyMatch`	Determines whether *any* stream elements match a specified condition; immediately terminates processing of the stream pipeline if an element matches.
`allMatch`	Determines whether *all* of the elements in the stream match a specified condition.

Exemplo de código

```
1  //            IntStreamOperations.java
2  // Demonstrating IntStream operations.
3  import java.util.Arrays;
4  import java.util.stream.IntStream;
5
6  public class IntStreamOperations
7  {
8     public static void main(String[] args)
9     {
10       int[] values = {3, 10, 6, 1, 4, 8, 2, 5, 9, 7};
11
12       // display original values
13       System.out.print("Original values: ");
14       IntStream.of(values)
15                .forEach(value -> System.out.printf("%d ", value));
16       System.out.println();
17
```

Exemplo de código

```
18          // count, min, max, sum and average of the values
19          System.out.printf("%nCount: %d%n", IntStream.of(values).count());
20          System.out.printf("Min: %d%n",
21             IntStream.of(values).min().getAsInt());
22          System.out.printf("Max: %d%n",
23             IntStream.of(values).max().getAsInt());
24          System.out.printf("Sum: %d%n", IntStream.of(values).sum());
25          System.out.printf("Average: %.2f%n",
26             IntStream.of(values).average().getAsDouble());
27
28          // sum of values with reduce method
29          System.out.printf("%nSum via reduce method: %d%n",
30             IntStream.of(values)
31                      .reduce(0, (x, y) -> x + y));
32
33          // sum of squares of values with reduce method
34          System.out.printf("Sum of squares via reduce method: %d%n",
35             IntStream.of(values)
36                      .reduce(0, (x, y) -> x + y * y));
```

Exemplo de código

```
37
38          // product of values with reduce method
39          System.out.printf("Product via reduce method: %d%n",
40             IntStream.of(values)
41                      .reduce(1, (x, y) -> x * y));
42
43          // even values displayed in sorted order
44          System.out.printf("%nEven values displayed in sorted order: ");
45          IntStream.of(values)
46                  .filter(value -> value % 2 == 0)
47                  .sorted()
48                  .forEach(value -> System.out.printf("%d ", value));
49          System.out.println();
50
51          // odd values multiplied by 10 and displayed in sorted order
52          System.out.printf(
53             "Odd values multiplied by 10 displayed in sorted order: ");
54          IntStream.of(values)
55                  .filter(value -> value % 2 != 0)
56                  .map(value -> value * 10)
57                  .sorted()
58                  .forEach(value -> System.out.printf("%d ", value));
59          System.out.println();
60
```

Exemplo de código

```
61          // sum range of integers from 1 to 10, exlusive
62          System.out.printf("%nSum of integers from 1 to 9: %d%n",
63             IntStream.range(1, 10).sum());
64
65          // sum range of integers from 1 to 10, inclusive
66          System.out.printf("Sum of integers from 1 to 10: %d%n",
67             IntStream.rangeClosed(1, 10).sum());
68       }
69    } // end class IntStreamOperations
```