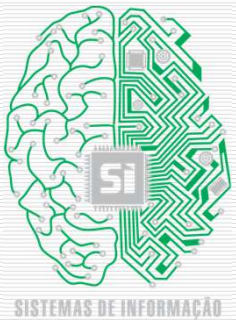
Expressões Lambda

Expressões Lambda



Expressões Lambda

- Tem como principal objetivo adicionar ao Java técnicas de linguagens funcionais, como Scala e LISP.
- A grande vantagem de funções lambda é diminuir a quantidade de código necessária para a escrita de algumas funções

Expressões Lambda

•Simplificando um pouco a definição, uma função lambda é uma função sem declaração, isto é, não é necessário colocar um nome, um tipo de retorno e o modificador de acesso.
•A ideia é que o método seja declarado no mesmo lugar em que será usado.

Expressões Lambda

•As funções lambda em Java tem a sintaxe definida como:

```
(argumento) -> (corpo)
```

Expressões Lambda

•Exemplos de funções lambda em Java.

```
(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
```

Expressões Lambda

•Uma função lambda pode ter nenhum ou vários parâmetros e seus tipos podem ser colocados ou podem ser omitidos, dessa forma, eles são inferidos pelo Java.
•A função lambda pode ter nenhum ou vários comandos:
•se a mesma tiver apenas um comando as chaves não são obrigatórias e a função retorna o valor calculado na expressão;

Expressões Lambda

•Se a função tiver vários comandos, é necessário colocar as chaves e também o comando *return* - se nada for retornado, a função tem um retorno *void*.

Expressões Lambda

Collections & Streams

- As funções lambdas podem ser bastante utilizadas com as classes de coleções do Java;
- Nessas classes fazemos diversos tipos de funções que consistem basicamente em percorrer a coleção

Expressões Lambda

- No ato de percorrer, podemos fazer uma determinada ação, como por exemplo, imprimir todos os elementos da coleção, filtrar elementos da lista e buscar um determinado valor na lista

```
for(Integer n: list) {
    System.out.println(n);
}
```

Expressões Lambda

- Com as funções lambda é possível implementar a mesma funcionalidade com menos código, bastando chamar o método forEach, que é um método que espera uma função lambda como parâmetro.
- `list.forEach(n -> System.out.println(n));`

Expressões Lambda

- Dentro do código de uma função lambda é possível executar diversos comandos. como por exemplo, imprimir apenas os pares:

```
list.forEach(n -> {
    if (n % 2 == 0) {
        System.out.println(n);
    }
});
```

Expressões Lambda

- Funções lambda podem ser utilizadas também para a ordenação:
- Considere uma lista de objetos Pessoa que possua o atributo Nome:

```
Collections.sort(listPessoas, (Pessoa
pessoa1, Pessoa pessoa2) ->
pessoa1.getNome().compareTo(pessoa2.g
etNome()));
```

Expressões Lambda

- Outra funcionalidade importante é o método stream() que foi acrescentado a interface Collection.
- Uma vez que Collection é a interface “pai” de todas as coleções, todas elas herdam então esse novo método:

Expressões Lambda

- Imagine uma lista de Pessoas com nome, Idade e Sexo

```
List<Pessoa> pessoas =
umMetodoQueRetornaPessoas();
Stream<Pessoa> stream =
pessoas.stream(); //criando um
stream de Pessoa
```

Expressões Lambda

- A interface Stream não é apenas mais um tipo de coleção.
- Um Stream é uma abstração de um “fluxo de dados”, para permitir exclusivamente manipulações e transformações sobre esses dados.

Expressões Lambda

- Diferente das coleções que conhecemos, um Stream não permite por exemplo que você acesse diretamente seus elementos (para isso tem que transformar o Stream em uma Collection novamente).

Expressões Lambda

- Filtros: Com expressões lambda também é possível filtrar elementos de uma coleção de objetos criando para isso um stream de dados

```
List<Pessoa> maioresTrinta =
pessoas.stream().filter(p ->
p.getIdade() >
30).collect(Collectors.toList());
```

Expressões Lambda

- Retornando lista de pessoas que iniciam com “E”

```
List<Pessoa> nomesIniciadosE =
pessoas.stream().filter(p ->
p.getNome().startsWith("E")).collect(
Collectors.toList());
```

Expressões Lambda

- Retornando operações:

```
long count =
pessoas.stream().filter(pessoa ->
pessoa.getSexo().equals("F")).count();
```

Expressões Lambda

- Retornando operações:

```
//pessoa mais nova com min()
Optional<Pessoa> nova = pessoas.stream()
        .min((p1, p2) ->
p1.getIdade().compareTo(p2.getIdade()));
```

Expressões Lambda

- Retornando operações:

```
//mais velha com max()
Optional<Pessoa> velha = pessoas.stream()
        .max((p1, p2) ->
p1.getIdade().compareTo(p2.getIdade()));
//imprimir as idades no console
System.out.println(nova.get().getIdade());
System.out.println(velha.get().getIdade());
```