DIGITAL
INNOVATION
ONE

Aula 4

Java Streams!

Collections

DIGITAL INNOVATION ONE

Objetivos

1. Classe Anônima
2. Functional Interface
3. Lambda
4. Method Reference
5. Stream API

Classe Anônima

A classe anônima em Java é uma classe não recebeu um nome e é tanto declarado e instanciado em uma única instrução. Você deve considerar o uso de uma classe anônima sempre que você precisa para criar uma classe que será instanciado apenas uma vez. Fonte

```
List<Gato> meusGatos = new ArrayList<>(){{
    add(new Gato( nome: "Jon",  idade: 12,  cor: "preto"));
    add(new Gato( nome: "Simba",  idade: 6,  cor: "tigrado"));
    add(new Gato( nome: "Jon",  idade: 18,  cor: "amarelo"));
}};
meusGatos.sort(new ComparatorIdade());

class ComparatorIdade implements Comparator<Gato> {
    @Override
    public int compare(Gato g1, Gato g2) {
        return Integer.compare(g1.getIdade(), g2.getIdade());
    }
}
```

Sem Classe anônima

```
List<Gato> meusGatos = new ArrayList<>(){{
    add(new Gato( nome: "Jon",  idade: 12,  cor: "preto"));
    add(new Gato( nome: "Simba",  idade: 6,  cor: "tigrado"));
    add(new Gato( nome: "Jon",  idade: 18,  cor: "amarelo"));
}};

meusGatos.sort(new Comparator<Gato>() {
    @Override
    public int compare(Gato g1, Gato g2) {
        return Integer.compare(g1.getIdade(), g2.getIdade());
    }
});
```

Com Classe anônima

DIGITAL INNOVATION ONE

Functional Interface

Qualquer interface com um SAM (Single Abstract Method) é uma interface funcional e sua implementação pode ser tratada como expressões lambda. Fonte

- Comparator
- Consumer
- Function
- Predicate

```java
@FunctionalInterface
public interface Comparator<T> {
    @Contract(pure = true) int compare(T var1, T var2);
```

Com Annotation FunctionalInterface

```java
public interface ActionListener extends EventListener {
    void actionPerformed(ActionEvent var1);
}
```

Sem Annotation FunctionalInterface

DIGITAL INNOVATION ONE

Lambda

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. As funções lambda em Java tem a sintaxe definida como (argumento) -> (corpo). Fonte

```
List<Gato> meusGatos = new ArrayList<>(){{
    add(new Gato( nome: "Jon",  idade: 12,  cor: "preto"));
    add(new Gato( nome: "Simba",  idade: 6,  cor: "tigrado"));
    add(new Gato( nome: "Jon",  idade: 18,  cor: "amarelo"));
}};

meusGatos.sort(Comparator.comparing(new Function<Gato, String>() {
    @Override
    public String apply(Gato gato) {
        return gato.getNome();
    }
}));
```

Sem Lambda Expressions

```
List<Gato> meusGatos = new ArrayList<>(){{
    add(new Gato( nome: "Jon",  idade: 12,  cor: "preto"));
    add(new Gato( nome: "Simba",  idade: 6,  cor: "tigrado"));
    add(new Gato( nome: "Jon",  idade: 18,  cor: "amarelo"));
}};

meusGatos.sort(Comparator.comparing((Gato gato) -> gato.getNome()));
```

Com Lambda Expressions

Reference Method

Method Reference é um novo recurso do Java 8 que permite fazer referência a um método ou construtor de uma classe (de forma funcional) e assim indicar que ele deve ser utilizado num ponto específico do código, deixando-o mais simples e legível . Para utilizá-lo, basta informar uma classe ou referência seguida do símbolo “::” e o nome do método sem os parênteses no final. Fonte

```
List<Gato> meusGatos = new ArrayList<>(){{
    add(new Gato( nome: "Jon",  idade: 12,  cor: "preto"));
    add(new Gato( nome: "Simba",  idade: 6,  cor: "tigrado"));
    add(new Gato( nome: "Jon",  idade: 18,  cor: "amarelo"));
}};

meusGatos.sort(Comparator.comparing((Gato gato) -> gato.getNome()));
```

Sem Reference Method

```
List<Gato> meusGatos = new ArrayList<>(){{
    add(new Gato( nome: "Jon",  idade: 12,  cor: "preto"));
    add(new Gato( nome: "Simba",  idade: 6,  cor: "tigrado"));
    add(new Gato( nome: "Jon",  idade: 18,  cor: "amarelo"));
}};
meusGatos.sort(Comparator.comparing(Gato::getNome));
```

Com Reference Method

DIGITAL INNOVATION ONE

Streams API

A Streams API traz uma nova opção para a manipulação de coleções em Java seguindo os princípios da programação funcional. Combinada com as expressões lambda, ela proporciona uma forma diferente de lidar com conjuntos de elementos, oferecendo ao desenvolvedor uma maneira simples e concisa de escrever código que resulta em facilidade de manutenção e paralelização sem efeitos indesejados em tempo de execução. Fonte

Fonte: tech.azi

DIGITAL INNOVATION ONE

Para saber mais

- **Implementando Collections e Streams com Java**

Instrutor: Wesley Fuchter

- **Desenvolvimento Avançado em Java**

Instrutor: João Paulo

- **Aprenda o que são estrutura de dados e algorítmos**

Instrutor: Bruno de Campos

DIGITAL INNOVATION ONE

REDES SOCIAIS


https://github.com/cami-la/curso-dio-intro-collections


https://www.linkedin.com/in/cami-la/


https://www.instagram.com/camimi_la

DIGITAL INNOVATION ONE

Dúvidas?

> Fórum do curso
> Comunidade online (discord)