AULA - 01

Entenda o que é paradigma funcional

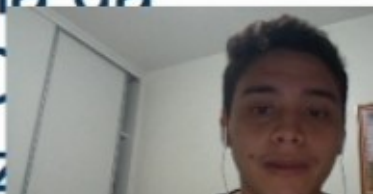
DIGITAL INNOVATION ONE

Objetivos da Aula

1. Entender o Paradigma Funcional no Java

2. Aprender como utilizar uma lambda e API Lambda do Java 8.

3. Entender paradigma da recursividade (Tail C Optimization e Memoriz



DIGITAL
INNOVATION
ONE

Requisitos Básicos

- ✓ Conceitos básicos de Java
- ✓ Orientação objeto
- ✓ Java Generics
- ✓ Collections: List e Set

DIGITAL
INNOVATION
ONE

Paradigma Funcional no Java

Programação funcional é o processo de construir software através de composição de funções puras, evitando compartilhamento de estados, dados mutáveis e efeitos colaterais. É declarativa ao invés de imperátiva, essa é uma definição de Eric Elliott.

DIGITAL INNOVATION ONE

Paradigma Funcional no Java

Paradigma Imperativo: E aquele que expressa o código através de comandos ao computador, nele é possível ter controle de estado dos objetos:

DIGITAL
INNOVATION
ONE

Paradigma Funcional no Java

```
class Imperativo {
    public static void main(String[] args) {
        int valor = 10;
        int resultado = valor * 3;
        System.out.println("O resutado é :: "+resultado);
    }
}
```

Imperativo.java

```java
public class Imperativo {
    public static void main(String[] args) {
        int valor = 10;
        int resultado = valor * 3; // instrução
        System.out.println("O resultado é :: " + resultado); // instrução
    }
}
```

```
> Task :Imperativo.main()
O resultado é :: 30
```

Paradigma Funcional no Java

Paradigma Funcional: Damos uma regra, uma declaração de como queremos que o programa se comporte.

DIGITAL INNOVATION ONE

Paradigma Funcional no Java

```
class Funcional {
    public static void main(String[] args) {
        UnaryOperator<Integer> calcularValorVezes3 = valor -> valor*3;
       int valor = 10;
        System.out.println("O resutado é :: "+calcularValorVezes3.apply(10));
    }
}
```

Imperativo.java | Funcional.java

```java
import java.util.function.UnaryOperator;

public class Funcional {
    public static void main(String[] args) {
        UnaryOperator<Integer> calcularValorVezes3 = valor -> valor*3;
        int valor = 10;
        System.out.println("O RESULTADO É :: " + calcularValorVezes3.apply( t: 10));
    }
}

```

```
> Task :Funcional.main()
O RESULTADO É :: 30
```

AULA - 02

Funções e imutabilidade em Paradigma Funcional

DIGITAL INNOVATION ONE

Paradigma Funcional no Java

Conceitos fundamentais da programação funcional.

Composição de funções: é criar uma nova função através da composição de outras. Por exemplo, vamos criar uma função que vai filtrar um array, filtrando somente os números pares e multiplicando por dois:

Paradigma Funcional no Java

Conceitos fundamentais da programação funcional.
Composição de funções:

```
class ComposicaoDeFuncoes {
    public static void main(String[] args) {
        int[] valores = {1,2,3,4};
        Arrays.stream(valores)
                .filter(numero -> numero % 2 == 0)
                .map(numero -> numero * 2)
                .forEach(numero -> System.out.println(numero));

    }
}
```

Imperativo.java | Funcional.java | ComposicaoDeFuncoes.java

```java
import java.util.Arrays;

public class ComposicaoDeFuncoes {
    public static void main(String[] args) {
        int [] valores = {1,2,3,4};
        Arrays.stream(valores)
                .filter(numero -> numero % 2 == 0)
                .map(numero -> numero * 2)
                .forEach(numero -> System.out.println("A SOMA É :: " + numero));
    }
}

```

```
> Task :ComposicaoDeFuncoes.main()
A SOMA É :: 4
A SOMA É :: 8
```

Imperativo.java × Funcional.java × ComposicaoDeFuncoes.java ×

```java
import java.util.Arrays;

public class ComposicaoDeFuncoes {
    public static void main(String[] args) {
        int [] valores = {1,2,3,4};
        // PARADIGMA FUNCIONAL
        Arrays.stream(valores)
                .filter(numero -> numero % 2 == 0)
                .map(numero -> numero * 2)
                .forEach(numero -> System.out.println("A SOMA É :: " + numero));

        // IMPERATIVO
        for(int i = 0; i < valores.length; i++){
            int valor = 0;
            if(valores[i] % 2 == 0){
                valor = valores[i] * 2;
                if(valor != 0){
                    System.out.println(valor);
                }
            }
        }
    }
}
```

```
> Task :ComposicaoDeFuncoes.main()
PARADIGMA FUNCIONAL :: 4
PARADIGMA FUNCIONAL :: 8
PARADIGMA IMPERATIVO: 4
PARADIGMA IMPERATIVO: 8
```

DIGITAL
INNOVATION
ONE

Paradigma Funcional no Java

Conceitos fundamentais da programação funcional.
Funções Puras: É chamada de pura quando invocada mais de uma vez produz exatamente o mesmo resultado.

Paradigma Funcional no Java

Conceitos fundamentais da programação funcional.
Funções Puras:

```
class FuncoesPuras {
    public static void main(String[] args) {
        BiPredicate<Integer,Integer> verificarSeEMaior =
                (parametro, valorComparacao) -> parametro > valorComparacao;

        System.out.println(verificarSeEMaior.test(13,12));
        System.out.println(verificarSeEMaior.test(13,12));
    }
}
```


FuncoesPuras.java

```java
import java.util.function.BiPredicate;

public class FuncoesPuras {
    public static void main(String[] args) {
        BiPredicate<Integer, Integer> verificarSeMaior =
                (parametro, valorComparacao) -> parametro > valorComparacao;

        System.out.println(verificarSeMaior.test( t: 13, u: 12));
        System.out.println(verificarSeMaior.test( t: 13, u: 12));
    }
}
```

```
> Task :FuncoesPuras.main()
true
true
```

DIGITAL INNOVATION ONE

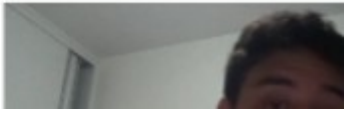
Paradigma Funcional no Java

Conceitos fundamentais da programação funcional.
Imutabilidade: Significa que uma vez que uma variável que recebeu um valor, vai possuir esse valor para sempre, ou quando criamos um objeto ele não pode ser modificado.

Paradigma Funcional no Java

Conceitos fundamentais da programação funcional.
Imutabilidade:

```
class Imultabilidade {
    public static void main(String[] args) {
        int valor = 20;
        UnaryOperator<Integer> retornarDobro = v -> v * 2;
        System.out.println(retornarDobro.apply(valor)); // retorna o dobro do valor
        System.out.println(valor); // valor nao sera alterado
    }
}
```

Imultabilidade.java

```java
import java.util.function.UnaryOperator;

public class Imultabilidade {
    public static void main(String[] args) {
        int valor = 20;
        UnaryOperator<Integer> retornarDodro = v -> v * 2;
        System.out.println(retornarDodro.apply(valor)); // RETORNA O DOBRO DO VALOR
        System.out.println(valor); // VALOR NÃO SERÁ ALTERADO;
    }
}

```

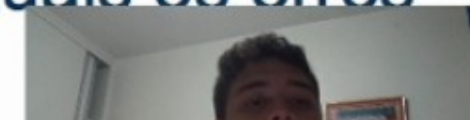
```
> Task :Imultabilidade.main()
40
20
```

Paradigma Funcional no Java

Imperativo x Declarativo

É muito comum aprender a programar de forma imperativa, onde mandamos alguém fazer algo. Busque o usuário 15 no banco de dados. Valide essas informações do usuário.

Na programação funcional tentamos programar de forma declarativa, onde declaramos o que desejamos, sem explicitar como será feito. Qual o usuário 15? Quais os erros dessas informações?



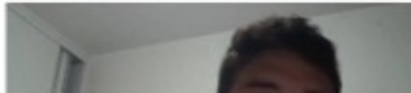
DIGITAL INNOVATION ONE

Exercício final

1. E aquele que expressa o código através de comandos ao computador, nele é possível ter controle de estado dos objetos, de acordo com a afirmação. Qual alternativa representa esse paradigma:

a) ( ) Declarativo
b) ( ) Imutabilidade
c) ( ) Funcional
d) ( ) Recursividade
e) ( ) Imperativo

RESPOSTA: e) Imperativo



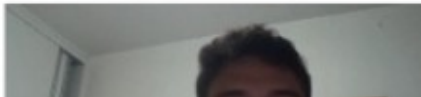
DIGITAL INNOVATION ONE

Exercício final

2. Damos uma regra, uma declaração de como queremos que o programa se comporte, de acordo com a afirmação. Qual alternativa representa esse paradigma:

a) ( ) Composição de funções
b) ( ) Imutabilidade
c) ( ) Funcional
d) ( ) Recursividade
e) ( ) Imperativo

RESPOSTA: c) Funcional



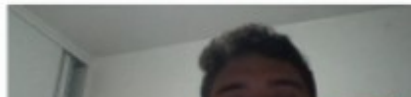
DIGITAL INNOVATION ONE

Exercício final

3. Uma vez que uma variável que recebe um valor, esta vai possuir esse valor para sempre, ou quando criamos um objeto ele não pode ser modificado, de acordo com a afirmação. Qual alternativa representa esse paradigma:

a) ( ) Composição de funções
b) ( ) Imutabilidade
c) ( ) Funcional
d) ( ) Recursividade
e) ( ) Imperativo

RESPOSTA: b) Imutabilidade



AULA - 03

Lambda no Java

DIGITAL
INNOVATION
ONE

Lambda no Java

Os lambdas obedecem o conceito do paradigma funcional, com eles podemos facilitar legibilidade do nosso código, alem disso com a nova API Funcional do Java podemos ter uma alta produtividade para lidar com objetos.
Primeiramente, devemos entender o que são interfaces funcionais.

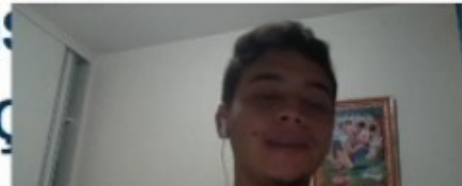
DIGITAL INNOVATION ONE

Lambda no Java

Interfaces funcionais- São interfaces que possuem apenas um método abstrato. Exemplo:

```
public interface Funcao {
    String gerar(String valor);
}
```

Geralmente as interfaces funcionais possuem uma anotação em nível de classe(@FunctionalInterface), para forçar o compilador a apontar um erro se a interface não estiver do acordo com as regras de uma interface funcional. Esta anotação não obrigatória, pois o compilador cons
reconhecer uma interface em tempo de compila



Lambda no Java

Antes do Java 8, se quiséssemos implementar um comportamento especifico para uma única classe deveríamos utilizar uma classe anônima para implementar este comportamento.

DIGITAL INNOVATION ONE

Lambda no Java

Exemplo com a interface Função(definida no slide 23):

```
public static void main(String[] args) {
    Funcao colocarPrefixoSenhorNaString = new Funcao() {
        @Override
        public String gerar(String valor) {
            return "Sr. "+valor;
        }
    };
    System.out.println(colocarPrefixoSenhorNaString.gerar("Joao"));
}

@FunctionalInterface
interface Funcao {
    String gerar(String valor);
}
```

DIGITAL
INNOVATION
ONE

Lambda no Java

Agora que sabemos como se define uma interface funcional podemos, aprender como se define uma lambda.
Estrutura de uma lambda:

InterfaceFuncional nomeVariavel = parametro → logica

Para entender melhor utilizaremos o exemplo da Função.

DIGITAL INNOVATION ONE

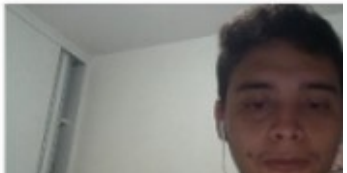
Lambda no Java

Vimos que a sintaxe fica bastante verbosa e o código fica bastante confuso utilizando esta implementação, agora escreveremos exatamente o mesmo código utilizando o lambda da interface Função:

```
class FuncaoComLambda {
    public static void main(String[] args) {
        Funcao colocarPrefixoSenhorNaString = valor -> "Sr. "+valor;
        System.out.println(colocarPrefixoSenhorNaString.gerar("Joao"));
    }
}

@FunctionalInterface
interface Funcao {
    String gerar(String valor);
}
```

A partir do Java 1.8



FuncaoComLambda.java

```java
package lambdas;

public class FuncaoComLambda {
    public static void main(String[] args) {
        Funcao colocarPrefixoSenhorNaString = valor -> "Sr. " + valor;
        System.out.println(colocarPrefixoSenhorNaString.gerar( valor: "João"));
    }
}
@FunctionalInterface
interface Funcao{
    String gerar(String valor);
}

```

```
> Task :FuncaoComLambda.main()
Sr. João
```

Lambda no Java

Bastante atenção !!!

- Quando uma lambda possui apenas uma instrução no corpo de sua lógica não é necessário o uso de chaves.

Exemplo:

```
Funcao colocarPrefixoSenhorNaString = valor -> "Sr. "+valor;
```

DIGITAL
INNOVATION
ONE

Lambda no Java

Bastante atenção !!!
Se a função possui mais de uma instrução DEVEMOS utilizar chaves e alem disso deve explicitar o retorno se o retorno for diferente de void. Exemplo:

```
Funcao colocarPrefixoSenhorNaString = valor -> {
   String valorComprefixo = "Sr. "+valor;
   String valorComPrefixoEPontoFinal = valorComprefixo+".";
   return valorComPrefixoEPontoFinal;
};
```

DIGITAL INNOVATION ONE

Exercício final

4. Qual é a sintaxe base de uma lambda?

a) ( ) Tipo nomeVariavel = parametro → logica

b) ( ) Tipo nomeVariavel = valor;

c) ( ) TipoAbstrato nomeVariavel = valor

d) (x) InterfaceFuncional nomeVariavel = parametro → logica

RESPOSTA: d)

e) ( ) TipoEnum nomeVariavel = parametro → logica

DIGITAL INNOVATION ONE

Exercício final

RESPOSTA: d)

5. Quando devemos utilizar chaves em um lambda?

a) ( ) Quando o mesmo possui apenas uma instrução
b) ( ) Quando utilizamos a referencia do método
c) ( ) Quando utilizamos a interface Runnable
d) (x) Quando o mesmo possui mais de uma instrução
e) ( ) Quando o mesmo não possui nenhuma instrução

DIGITAL INNOVATION ONE

Exercício final

6. Qual das alternativas a seguir e uma declaração de um lambda.

a) ( ) Funcao a = "2";
b) ( ) Funcao a => "2";
c) ( ) Funcao a = a => "2";
d) ( ) Funcao a -> "2";
e) (x) Funcao a = valor -> "2";

RESPOSTA: e)

DIGITAL INNOVATION ONE

Exercício final

7. Antes do Java 8, qual era a estratégia utilizada para implementação de interface em uma classe especifica.

a) (x)) Classe Anônima;
b) ( ) Declaração de uma outra interface;
c) ( ) Enum;
d) ( ) Classe Abstrata;
e) ( ) Encapsulamento;

RESPOSTA: a)

AULA - 04

Recursividade em Java

DIGITAL INNOVATION ONE

Recursividade

Na recursividade, uma função chama a si mesma repetidamente, até atingir uma condição de parada.
No caso de Java, um método chama a si mesmo, passando para si objetos primitivos. Cada chamada gera uma nova entrada na pilha de execução, e alguns dados podem ser disponibilizados em um escopo global ou local, através de parâmetros em um escopo global ou local.

Recursividade

Recursividade tem um papel importante em programação funcional, facilitando que evitemos estados mutáveis e mantenhamos nosso programa mais declarativo, e menos imperativo.

DIGITAL INNOVATION ONE

Recursividade

Exemplo do calculo fatorial de forma recursiva:

```
class FatorialRecursivo {
    public static void main(String[] args) {
        System.out.println(fatorial(5));
    }


    public static int fatorial( int value ) {
        if ( value == 1 ) {
            return value;
        } else {
            return value * fatorial((value -1));
        }
     }
}
```

FatorialRecursivo.java

```java
package recursividade;

public class FatorialRecursivo {
    public static void main(String[] args) {
        System.out.println(fatorial( value: 5));
    }

    private static int fatorial(int value) {
        if( value == 1 ){
            return value;
        }else{
            return value * fatorial((value -1));
        }
    }
}

```

```
> Task :FatorialRecursivo.main()
120
```

Como a recursividade funciona:

Recursividade

Explicação Recursividade:

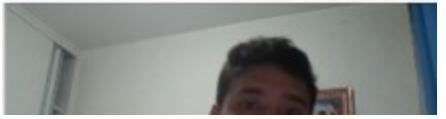
```
(fatorial(5))
(5 * (fatorial(4)))
(5 * (4 * (fatorial(3))))
(5 * (4 * (3 * (fatorial(2)))))
(5 * (4 * (3 * (2 * (fatorial(1))))))
(5 * (4 * (3 * (2 * (1 * (fatorial(0)))))))
(5 * (4 * (3 * (2 * (1 * (1))))))
(5 * (4 * (3 * (2 * (1)))))
(5 * (4 * (3 * (2))))
(5 * (4 * (6)))
(5 * (24))
120
```

DIGITAL INNOVATION ONE

Recursividade

Tail Call (Recursividade em cauda) : Recursão em cauda é uma recursão onde não há nenhuma linha de código após a chamada do próprio método e, sendo assim, não há nenhum tipo de processamento a ser feito após a chamada recursiva.

Obs: a JVM não suporta a recursão em cauda, ele lança um estouro de pilha (StackOverFlow)

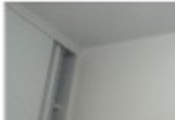


DIGITAL INNOVATION ONE

Recursividade

Fatorial com o Tail Call:

```
class FatorialTailCall {
    public static void main(String[] args) {
        System.out.println(fatorialA(5));
    }
    public static double fatorialA( double valor ) {
        return fatorialComTailCall(valor,1);
    }
    public static double fatorialComTailCall(double valor, double numero){
        if (valor == 0){
            return numero;
        }
        return fatorialComTailCall(valor-1,numero*valor);
    }
}
```



FatorialTailCall.java

```java
package recursividade;

public class FatorialTailCall {
    public static void main(String[] args) {
        System.out.println(fatorialA( valor: 5));
    }

    private static double fatorialA(double valor) {
        return fatorialComTailCall(valor, numero: 1);
    }

    private static double fatorialComTailCall(double valor, double numero) {
        if( valor == 0 ) {
            return numero;
        }
        return fatorialComTailCall( valor: valor-1, numero: numero*valor);
    }
    // 5 * 4 * 3 * 2 * 1

}
```

```
> Task :FatorialTailCall.main()
120.0
```

DIGITAL INNOVATION ONE

Recursividade

Explicação Tail Call:

```
fatorialA(5,1)
fatorialA(4,5)
fatorialA(3,20)
fatorialA(2,60)
fatorialA(1,120)
fatorialA(0,120)
120
```

DIGITAL INNOVATION ONE

Recursividade

Memoization : é uma técnica de otimização que consiste no cache do resultado de uma função, baseado nos parâmetros de entrada. Com isso, nas seguintes execuções conseguimos ter uma resposta mais rápida

DIGITAL INNOVATION ONE

Exercício final

8. Qual o problema que pode ocorrer ao utilizar recursividade?

a) ( ) Deadlock
b) ( ) Starvation
c) ( ) Concorrência
d) (X) Estouro de pilha (StackOverFlow)
e) ( ) Erro de compilação.

RESPOSTA: d)

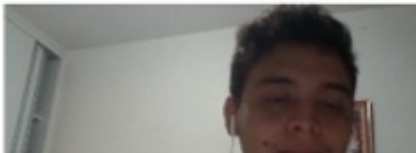
DIGITAL INNOVATION ONE

Exercício final

9. É uma técnica de otimização que consiste no cache do resultado de uma função, baseada nos parâmetros de entrada, a partir desta afirmação, julgue o item correto.

a) ( ) Memoization
b) ( ) Taill Call
c) ( ) Lambda
d) (X) Paradigma Imperativo
e) ( ) Imutabilidade

RESPOSTA: a)



DIGITAL INNOVATION ONE

Exercício final

10. É uma recursão onde não há nenhuma linha de código após a chamada do próprio método e, sendo assim, não há nenhum tipo de processamento a ser feito após a chamada recursiva, a partir desta afirmação, julgue o item correto.

a) ( ) Memoization
b) (X) Taill Call
c) ( ) Lambda
d) ( ) Paradigma Imperativo
e) ( ) Imutabilidade