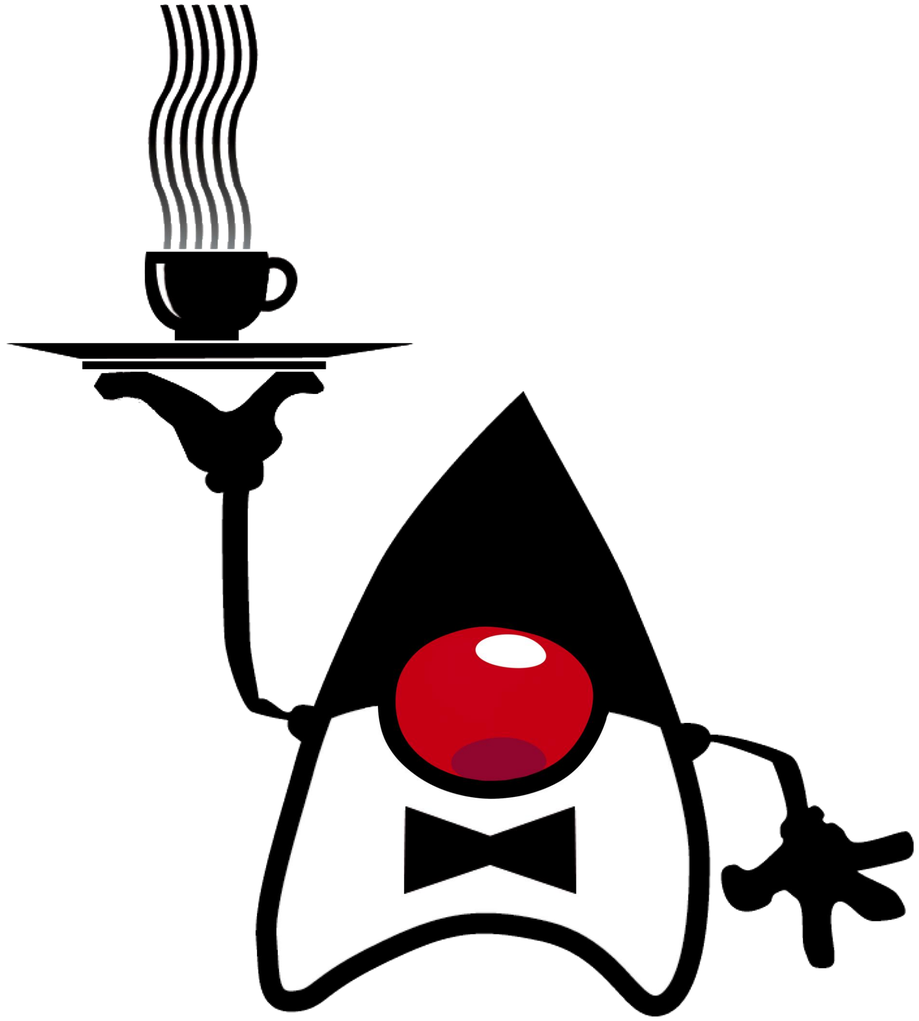
Trilha Java

Encontro 25 – Programação Funcional:

Expressões Lambda



vnt/school
powered by venturus

Recapitulação

1. Generics
2. hashCode e equals
3. Set
4. Map

vnt/school
powered by venturus

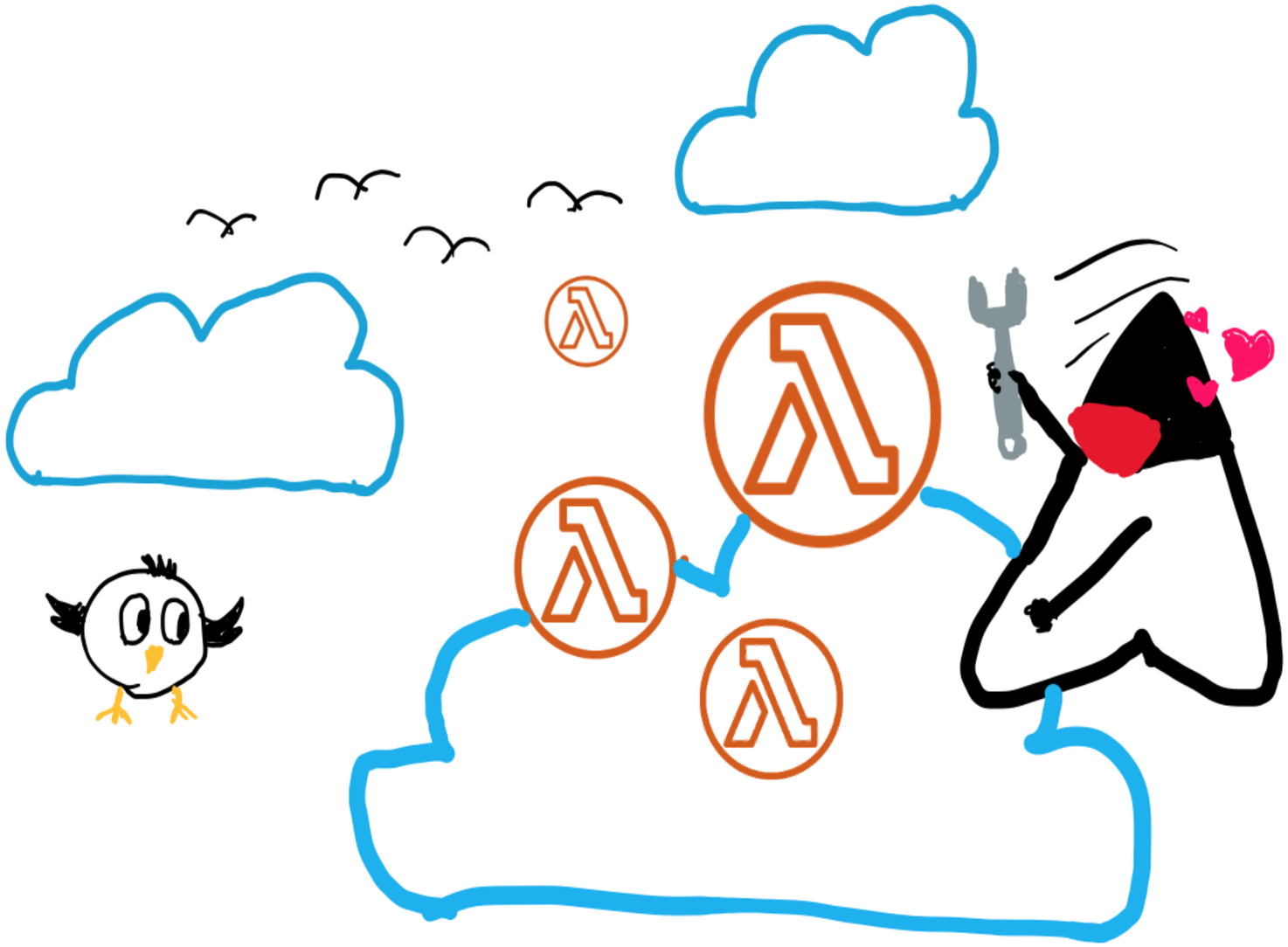
Agenda

1. Interface Funcional
2. Expressões Lambda
3. Predicate
4. Consumer
5. Function
6. Exemplos

vnt/school
powered by venturus

Programação Funcional

Expressões Lambda



vnt/school

powered by venturus

Programação Funcional

Suponha uma classe **Produto** com os atributos **name** e **price**.

Produto
- String **name** - Double **price**

Podemos **implementar a comparação** de produtos por meio da implementação da **interface Comparable**.

Produtos	
TV	900.0
Notebook	1200.0
Tablet	400.0

vnt/school
powered by venturus

Comparator objeto: interface comparable

Exemplo 1:

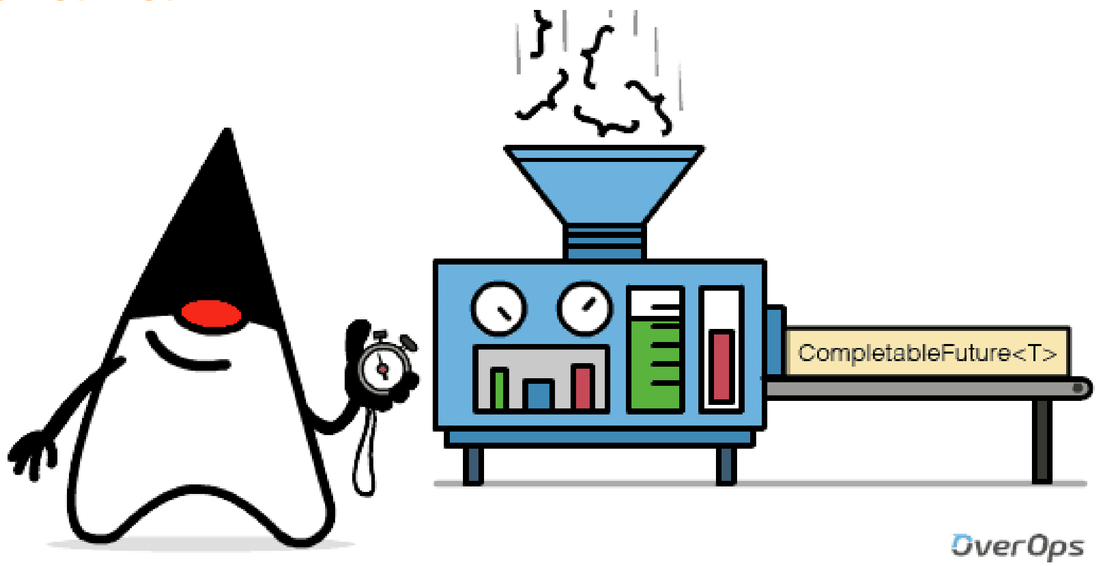
public class Produto implements Comparable<Produto>

```
public class Exemplo1 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();

        list.add(new Produto("TV", 900.00));
        list.add(new Produto("Notebook", 1200.00));
        list.add(new Produto("Tablet", 450.00));

        Collections.sort(list);

        for (Produto p : list) {
            System.out.println(p);
        }
```

Programação Funcional

Entretanto, desta forma nossa classe não fica **fechada para alteração**: se o critério de comparação mudar, precisaremos **alterar a classe Produto**.

Podemos então usar o default method "**sort**" da interface List:
default void sort(Comparator c)



vnt/school
powered by venturus

Programação Funcional

Comparator objeto de **classe separada**

Comparator objeto de **classe anônima**

Comparator objeto de **expressão lambda com chaves**

Comparator objeto de **expressão lambda sem chaves**

Comparator **expressão lambda "direto no argumento"**

Comparator objeto: classe separada

Exemplo 2:

Além da classe **Produto**, deve ser criada de forma separada a classe **Comparacao.**

public class Comparacao implements Comparator<Produto>

```
public class Exemplo2 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("TV", 900.00));
        list.add(new Produto("Notebook", 1200.00));
        list.add(new Produto("Tablet", 450.00));

        list.sort(new Comparacao());

        for (Produto p : list) {
            System.out.println(p);
        }
```

vnt/school
powered by venturus

Comparator objeto: classe anônima

Exemplo 3:

Uma **classe anônima** é criada no programa principal.

A variável **comp** é instanciada como um objeto do tipo **Comparator**.

```
public class Exemplo3 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("TV", 900.00));
        list.add(new Produto("Notebook", 1200.00));
        list.add(new Produto("Tablet", 450.00));
        Comparator<Produto> comp = new Comparator<Produto>(){
            @Override
            public int compare(Produto p1, Produto p2) {
               return p1.getName().toUpperCase().compareTo(p2.getName().toUpperCase());
            }};
        list.sort(comp);
        for (Produto p : list) {
            System.out.println(p);
        }
    }
```

vnt/school
powered by venturus

Comparator objeto: expressão lambda com chaves

Exemplo 4:

Podemos deixar o programa **menos verboso**. Basta usar **Expressões Lambda**.

```
public class Exemplo4 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("TV", 900.00));
        list.add(new Produto("Notebook", 1200.00));
        list.add(new Produto("Tablet", 450.00));

        Comparator<Produto> comp = (p1, p2) -> {
        return p1.getName().toUpperCase().compareTo(p2.getName().toUpperCase());
        };

        list.sort(comp);
        for (Produto p : list) {
            System.out.println(p);
        }
    }
```

Comparator objeto: expressão lambda direto no argumento

Exemplo 4:

A forma mais simplifica é inserindo a **Expressão Lambda** , que é uma função anônima, **dentro do argumento**.

```
public class Exemplo4 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("TV", 900.00));
        list.add(new Produto("Notebook", 1200.00));
        list.add(new Produto("Tablet", 450.00));

        list.sort((p1, p2) -> p1.getName().toUpperCase().compareTo(p2.getName().toUpperCase()));

        for (Produto p : list) {
            System.out.println(p);
        }
```

vnt/school
powered by venturus

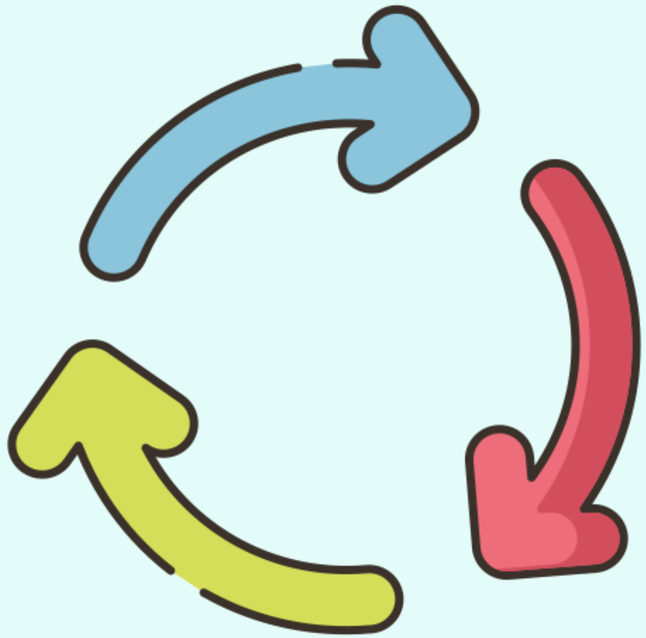
Programação Funcional

Resumindo:

* As funções possuem transparência referencial.

* Existe muita expressividade e o código é conciso.

* A tipagem é dinâmica e existe inferência de tipos.

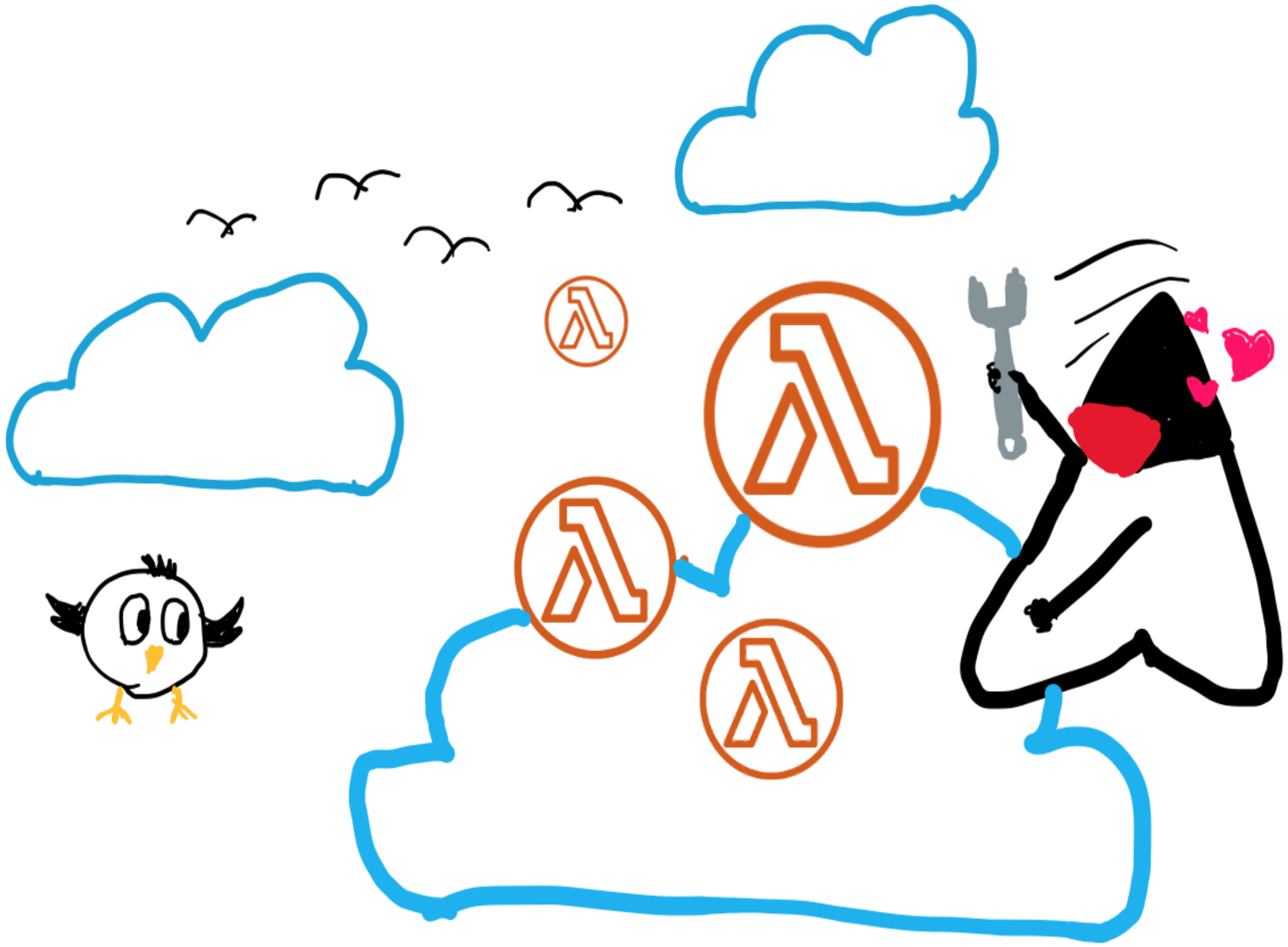


vnt/school

powered by venturus

Interface Funcional

Expressões Lambda

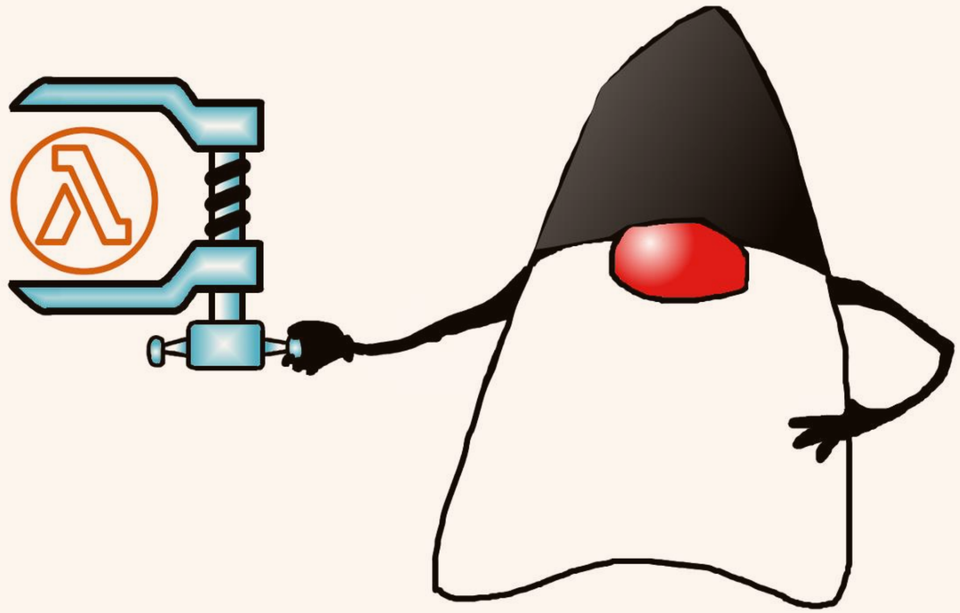


vnt/school

powered by venturus

Interface Funcional

É uma **interface** que possui um único **método abstrato**. Suas implementações serão tratadas como **expressões lambda**.



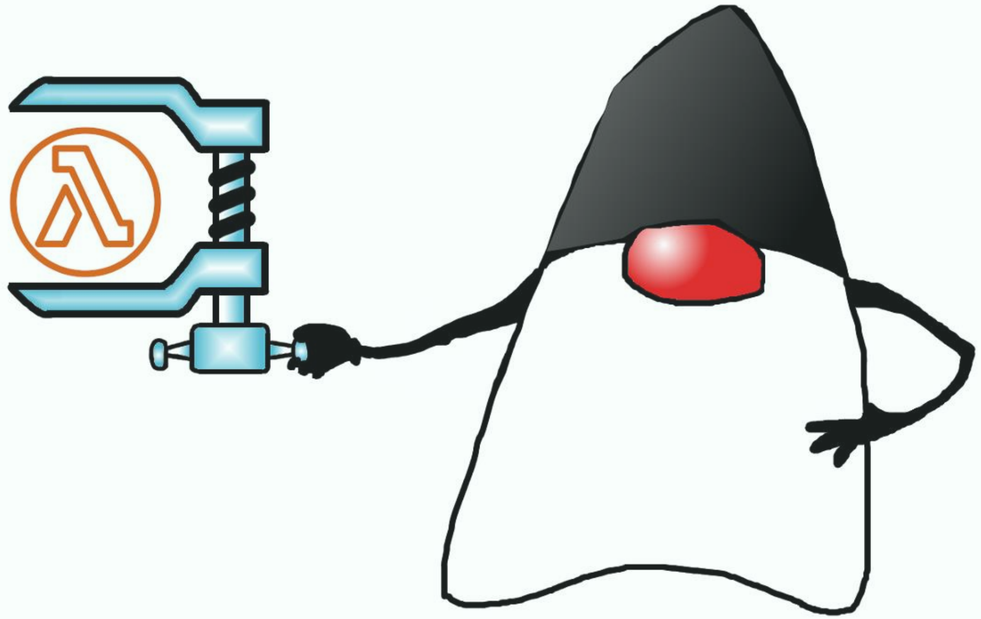
Em resumo, a **Expressão Lambda** será tratada com uma **interface funcional**. Exemplo feito anteriormente com a interface "**Comparacao**".

vnt/school
powered by venturus

Interface Funcional

Algumas outras **interfaces** funcionais comuns:

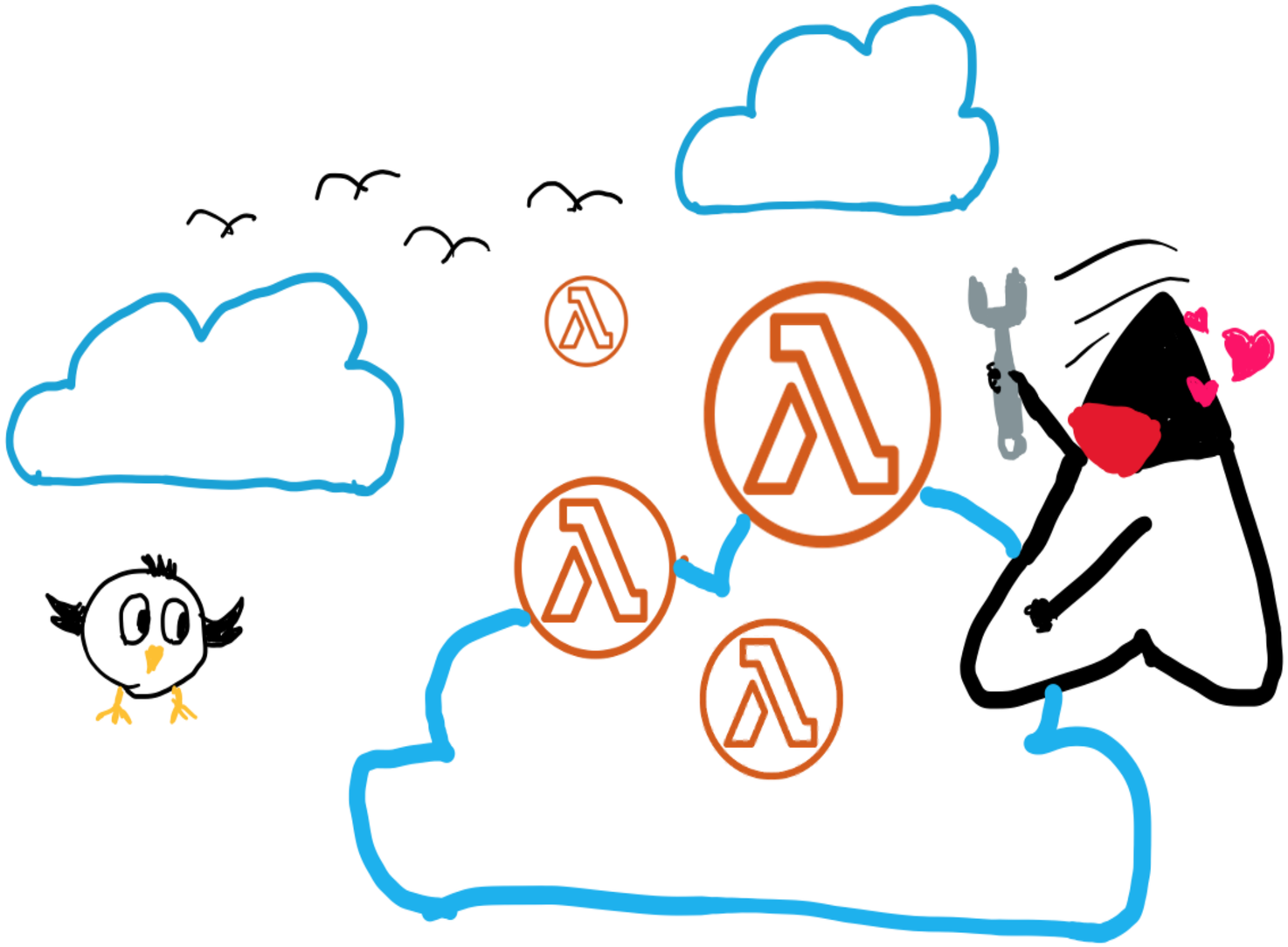
Predicate.
Consumer.
Function.



vnt/school
powered by venturus

Predicate:

Expressões Lambda



vnt/school

powered by venturus

Predicate

Exemplo 5:

A função **removeIf()**, remove da lista os produtos conforme a lógica dentro do argumento.

Veja que há **inferência** que **p** é uma **produto**, sem precisar explicitar.

No entanto, isso pode ser feito através de uma interface.

```
public class Exemplo5 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        list.removeIf(p -> p.getPrice() >= 100);

        for (Produto p : list) {
            System.out.println(p);
        }
```

vnt/school
powered by venturus

Predicate

Exemplo 6:

Uma **instância** da classe **ProdutoPredicate** foi inserida no argumento da função **removeIf()**.

Essa interface **ProdutoPredicate** constrói a **lógica/critério** que deve ser executado, quando chamado no código.

```
public class Exemplo6 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        list.removeIf(new ProdutoPredicate() );

        for (Produto p : list) {
            System.out.println(p);
        }
```

vnt/school
powered by venturus

Predicate

Outras formas de implementar o Predicate:

Reference method com método estático.

Expressão lambda declarada.

Expressão lambda inline.

vnt/school

powered by venturus

Predicate: Reference method

Exemplo 7:

É uma **referência para método** utilizando **método estático**.

Um **método estático** foi criado dentro da classe **Produto** e referenciada no **programa principal**.

```
public class Exemplo7 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        list.removeIf(Produto::StaticProdutoPredicate );

        for (Produto p : list) {
            System.out.println(p);
        }
```

vnt/school
powered by venturus

Predicate: Expressão lambda declarada

Exemplo 8:

Uma **expressão lâmbida** é criada entro do **programa principal**.

A variável **pred** é criada para o tipo **Predicate** e possui a lógica do que deve ser executado.

```
public class Exemplo8 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        Predicate<Produto> pred = p -> p.getPrice() >= 100;
        list.removeIf(pred);

        for (Produto p : list) {
            System.out.println(p);
        }
```

vnt/school
powered by venturus

Predicate: Expressão lambda inline

Exemplo 8:

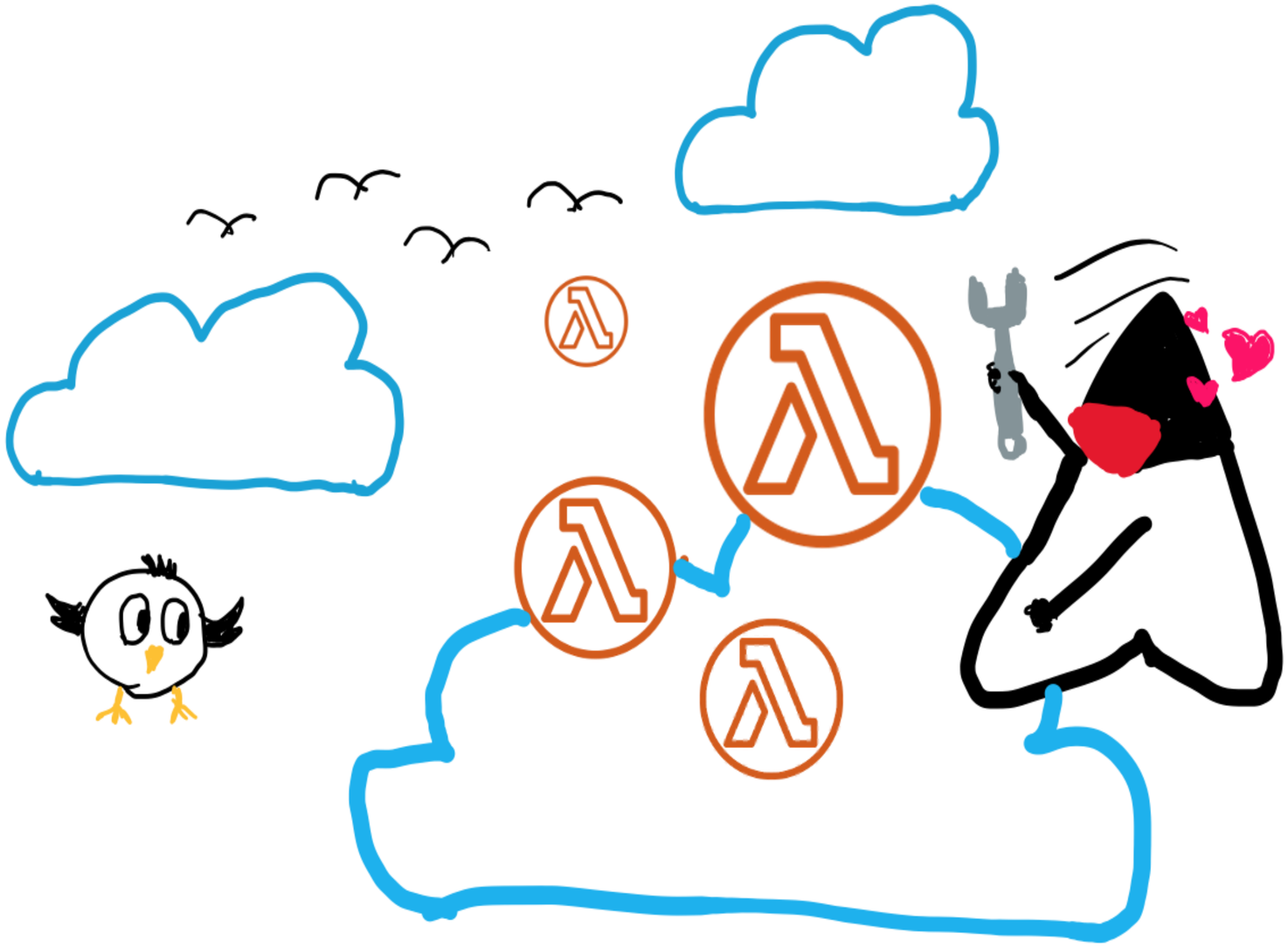
A **expressão lâmbida** é inserida diretamente no lugar da variável.

Veja que há **inferência** que **p** é uma **produto**, sem precisar explicitar.

```
public class Exemplo8 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        list.removeIf(p -> p.getPrice() >= 100);

        for (Produto p : list) {
            System.out.println(p);
        }
```

Consumer: forEach

Expressões Lambda



vnt/school
powered by venturus

Consumer: forEach

É uma interface funcional, que também pode ser implementada de diversas formas:

Implementação do Consumer.
Reference method com método estático.
Expressão lambda declarada.
Expressão lambda inline.

Consumer: forEach

Suponha ter que fazer um programa que, a partir de uma **lista de produtos**, aumente o preço dos produtos em **10%**. Utilize os produtos/código das aulas anteriores. Além disso, será usado o método **forEach:**

```
list.forEach(new PrecoAtualizado());
```

E um Consumer deve ser Implementado:

```
public class PrecoAtualizado implements Consumer<Produto>
```

Produto
- String **name** - Double **price**

Produtos	
TV	900.0
Mouse	50.0
Tablet	350.50
HD Case	80.90

vnt/school
powered by venturus

Consumer: implementação consumer

Exemplo 9:

A classe **PrecoAtualizado** é instanciada dentro do **forEach**.

O método **println** também pode ser executado dentro do **forEach**.

```
public class Exemplo9 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        list.forEach(new PrecoAtualizado());
        list.forEach(System.out::println);
    }
```

vnt/school
powered by venturus

Atividade:

Reproduza o código anterior e aplique os 10% nos produtos, utilizando o caminhos citados abaixo:

Reference method com método estático.
Expressão lambda declarada.
Expressão lambda inline.

vnt/school
powered by venturus

Consumer: Reference method

Exemplo 10:

O método **staticPrecoAtualizado** é criado dentro da classe **Produto**.

Este método é chamado como **argumento** do **fotEach** no programa principal.

```
public class Exemplo10 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        list.forEach(Produto::staticPrecoAtualizado);
        list.forEach(System.out::println);
    }
```

vnt/school
powered by venturus

Consumer: Expressão lambda declarada

Exemplo 11:

Uma variável **cons** do tipo **Consumer** é criada.

A **expressão lambda** é desenvolvida.

```
public class Exemplo11 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        Consumer<Produto> cons = p -> p.setPrice(p.getPrice()*1.1);
        list.forEach(cons);
        list.forEach(System.out::println);
    }
```

vnt/school
powered by venturus

Consumer: Expressão lambda inline

A **expressão lambda** é passada diretamente dentro do **argumento** no **forEach**.

.

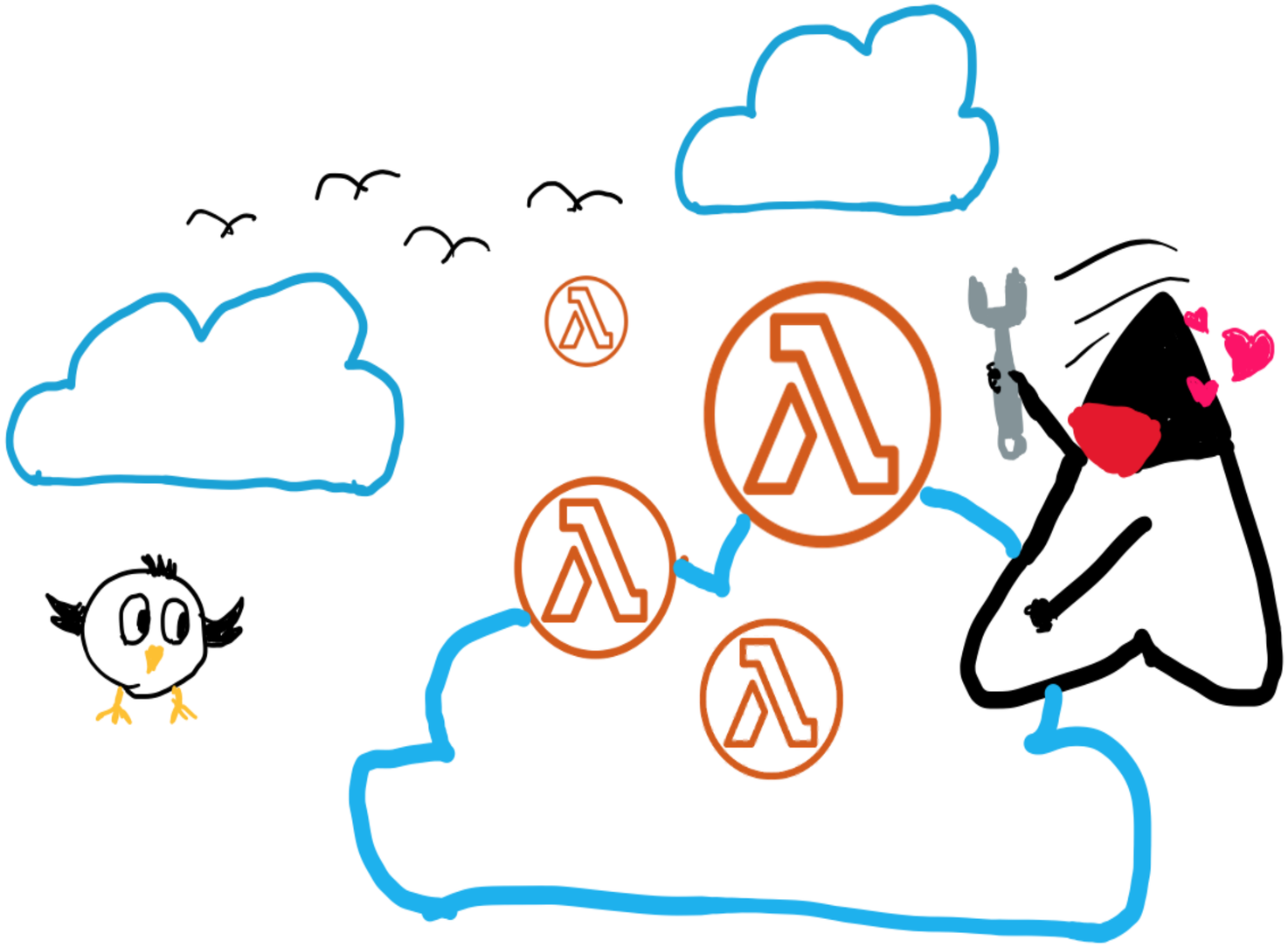
```
public class Exemplo11 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        list.forEach(p -> p.setPrice(p.getPrice()*1.1));
        list.forEach(System.out::println);
    }
```

vnt/school
powered by venturus

coffee
time

Function

Expressões Lambda



vnt/school

powered by venturus

Function

Representa uma **função** que recebe um argumento e produz um resultado. Assim, esta **interface funcional** abrange 2 genéricos, nomeadamente:

T: denota o tipo do argumento de entrada.

R: denota o tipo de retorno da função.

```
public interface Function<T,
R> {
    R apply(T t);
}
```

vnt/school
powered by venturus

Function

A **interface Function** consiste nos 4 métodos listados a seguir:

apply()
andThen()
compose()
identity()

vnt/school
powered by venturus

Function

Exemplo: Fazer um programa que, a partir de uma lista de produtos, gere uma nova lista contendo os nomes dos produtos em caixa alta.

Produtos	
Tv	900.00
Mouse	50.00
Tablet	350.50
HD Case	80.90

vnt/school
powered by venturus

Function

Nota sobre a função map:

A **função "map"** (não confunda com a estrutura de dados Map) é uma função que aplica uma função a todos elementos de uma **stream**.

Conversões:

List para stream: **.stream()**

Stream para List: **.collect(Collectors.toList())**

vnt/school

powered by venturus

Function: implementação Function

Exemplo 12:

A classe **UpperCaseName** é instanciada dentro da função **map()**.

A **list** é transformada em **stream** e depois transformada em **list** novamente.

```
public class Exemplo12 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        List<String> names = list.stream().map(new UpperCaseName()).collect(Collectors.toList());
        names.forEach(System.out::println);
    }
```

vnt/school

powered by venturus

Function: Reference method

Exemplo 13:

O método **UpperCaseName** é criado dentro da classe **Produto**.

Este método é chamado como **argumento** do **map()** no programa principal.

```
public class Exemplo13 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        List<String> names = list.stream().map(Produto::staticUpperCaseName)
                .collect(Collectors.toList());
        names.forEach(System.out::println);
    }
```

vnt/school
powered by venturus

Function: Expressão lambda declarada

Exemplo 14:

Uma variável **func** do tipo **Function** é criada.

A **expressão lambda** é desenvolvida em **func**.

```
public class Exemplo14 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        Function<Produto, String> func = p -> p.getName().toUpperCase();
        List<String> names = list.stream().map(func)
                .collect(Collectors.toList());
        names.forEach(System.out::println);
    }
```

vnt/school
powered by venturus

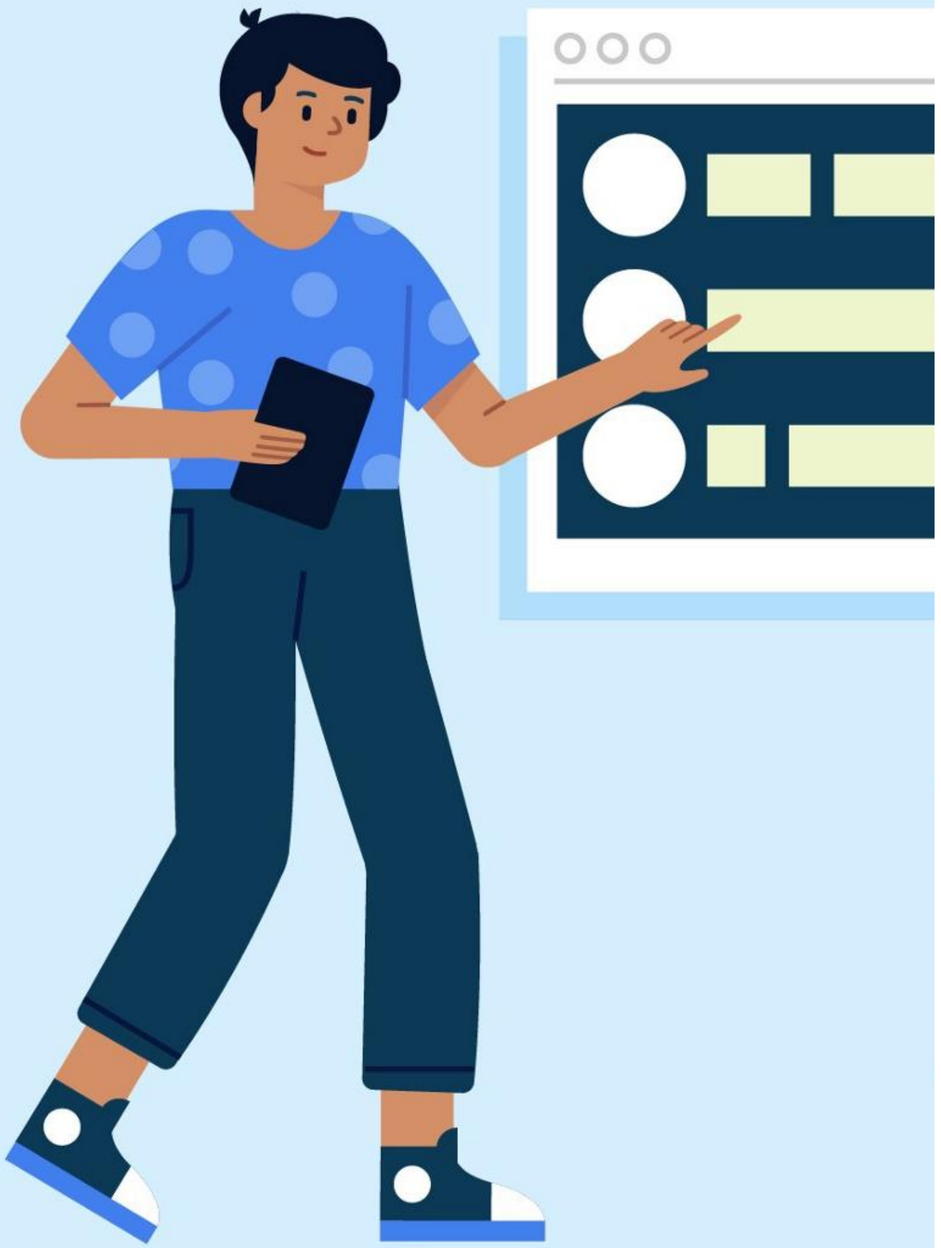
Function: Expressão lambda inline

A **expressão lambda** é passada diretamente dentro do **argumento** no **map()**.

.

```
public class Exemplo14 {
    public static void main(String[] args) {
        List<Produto> list = new ArrayList<>();
        list.add(new Produto("Tv", 900.00));
        list.add(new Produto("Mouse", 50.00));
        list.add(new Produto("Tablet", 350.50));
        list.add(new Produto("HD Case", 80.90));

        List<String> names = list.stream().map(p -> p.getName().toUpperCase())
                .collect(Collectors.toList());
        names.forEach(System.out::println);
    }
```


Review
e
Preview



Comunidade VNT

Dica de hoje

O link abaixo representa a página oficial da Oracle. Explore o conteúdo de Comparator e de List. Visite a página e busque mais informações sobre o tema.

Comparator:

Comparator (Java SE 10 & JDK 10) (oracle.com)

List:

List (Java SE 10 & JDK 10) (oracle.com)

Dica de hoje

O link abaixo representa a página oficial da Oracle. Explore o conteúdo de Predicate, Consumer e Function. Visite a página e busque mais informações sobre o tema.

Predicate:
Predicate (Java SE 10 & JDK 10) (oracle.com)

Consumer:
Consumer (Java SE 10 & JDK 10) (oracle.com)

Function:
Function (Java SE 10 & JDK 10) (oracle.com)

Referências

[1] A. Goldman, F. Kon, Paulo J. S. Silva; Introdução à Ciência da Computação com Java e Orientação a Objetos (USP). 2006. Ed. USP.
[2] Algoritmo e lógica de programação. Acessado julho/2022: https://visualg3.com.br/
[3] G. Silveira; Algoritmos em Java; Ed. Casa do Código.
[4] M. T. Goodrich, R. Tamassia; Estrutura de dados e algoritmos em Java. Ed Bookman. 2007.
[5] Algoritmo e lógica de programação. Acessado julho/2022: https://www.cursoemvideo.com/
[6] P. Silveira, R. Turini; Java 8 Pratico: lambdas, streams e os novos recursos da linguagem. Ed. Casa do Código.
[7] Linguagem Java: Curso acessado em agosto/2022: https://www.udemy.com/
[8] Linguagem Java: Curso acessado em setembro/2022: https://www.cursoemvideo.com/