ISPTEC

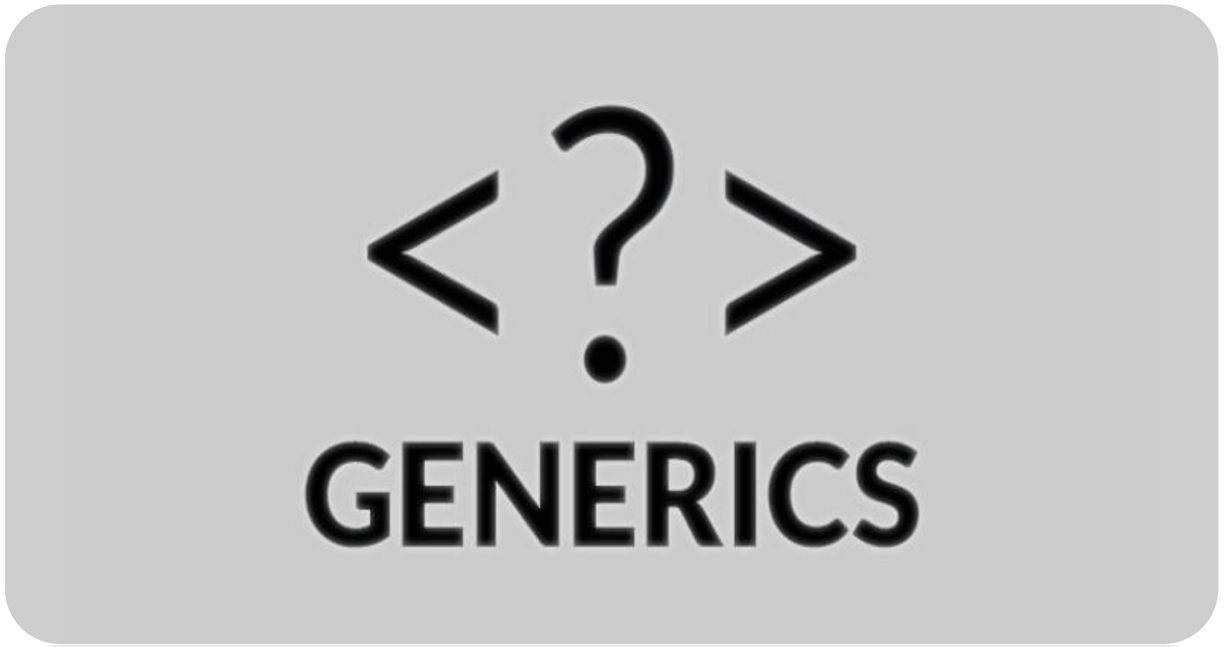
Instituto Superior Politécnico de Tecnologias e Ciências

Programação III

Autor: Sediangani Sofrimento
Área: DET

Sumário

- Problema
- Generics
- Classes Genéricas
- Métodos Genéricos
- Limitar o alcance de um Genérico
- Object Vs Generic



Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Problema 1

- Vamos supor que pretendemos criar uma **classe** que guarda um **inteiro** e que o imprima quando entendermos.

```
public class IntegerPrinter {

    Integer valueToPrint;

    public IntegerPrinter(Integer valueToPrint) {
        this.valueToPrint = valueToPrint;
    }

    public void print() {
        System.out.println(valueToPrint);
    }
}
```

```
public static void main(String[] args) {
    // TODO code application logic here

    IntegerPrinter printer = new IntegerPrinter(23);
    printer.print();
}
```

Qual seria o Output ?

23

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Problema 1

- E caso pretendessemos o mesmo comportamento da **classe** mas para o tipo **double**?
  - Não poderiamos utilizar a classe **IntegerPrinter** uma vez que essa guarda um valor **inteiro** e não **double**. Então basicamente teriamos de fazer uma cópia da classe anterior e alterar onde esta o **tipo inteiro** para o **tipo double.**

```
public class DoublePrinter {

    Double valueToPrint;

    public DoublePrinter(Double valueToPrint) {
        this.valueToPrint = valueToPrint;
    }

    public void print() {
        System.out.println(valueToPrint);
    }
}
```

```
public static void main(String[] args) {
    // TODO code application logic here

    DoublePrinter printer = new DoublePrinter(23.0);
    printer.print();
}
```

Qual seria o Output ?

23.0

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Problema 1

- E se pretendessemos o mesmo comportamento para o tipo float, char, String, Cao, Gato, Pessoa e etc?

- Fariamos copy paste da classe e alterar o tipo para o tipo de dados que pretendessemos ?

Generics

- Generics permite que tipos(classes e interfaces) sejam parâmetros na definição de **classes**, **interfaces** e **métodos**

- Permite o reuso de código para diferentes **tipos**

  - Sobrecarga de métodos, em geral, replica o mesmo código para tipos de dados diferentes

  - Evita o uso de **casting explícito** por parte do programador

- Um exemplo é a classe **ArrayList**
  - Quais tipos de objectos deve armazenar?

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Classes Genéricas


```java
public class Printer<T> {

    T valueToPrint;

    public Printer(T valueToPrint) {
        this.valueToPrint = valueToPrint;
    }

    public void print() {
        System.out.println(valueToPrint);
    }
}
```

```java
public static void main(String[] args) {
    Printer<Integer> integerPrinter = new Printer<>(23);
    integerPrinter.print();

    Printer<Double> doublePrinter = new Printer<>(23.0);
    doublePrinter.print();

    Printer<String> stringPrinter = new Printer<>("Ola Mundo");
    stringPrinter.print();
}
```

Qual seria o Output ?

23
23.0
Ola Mundo

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Classes Genéricas

- A classe **Printer** tornou-se uma **"classe Genérica"** ou **"parametrizada"** e vai poder tratar de qualquer **tipo dados**.
  - **public Printer<T>**
    - Para especificar o **parâmetro** podemos utilizar qualquer palavra desde que não seja uma **KeyWord.**
    - Contudo, por convenção, tipos são definidos com uma única letra maiúscula
      - E – Element (Java Collections)
      - K – Key
      - N – Number
      - T - Type

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Classes Genéricas


```
public class Printer<T> {

    T valueToPrint;

    public Printer(T valueToPrint) {
        this.valueToPrint = valueToPrint;
    }

    public void print() {
        System.out.println(valueToPrint);
    }
}
```

- O parâmetro é incluído entre o **diamante** <> logo após o nome da classe.
- O Tipo **T** pode ser qualquer tipo não primitivo

```
public class Box<T> { // T significa Tipo
    private T t;

    public void set(T t){
        this.t = t;
    }
    public T get() {
        return t;
    }
}
```

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Classes Genéricas

- A declaração e a instanciação de um tipo genérico deve especificar qual o tipo desejado
- Similar à chamada de um método ou constructor para o qual passamos parâmetros
  - A diferença é que em **Generics**, o parâmetro é um tipo(classe ou interface)

```
Box<Integer> integerBox = new Box<Integer>();
```

ou

```
Box<Integer> integerBox = new Box<>();
```

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Métodos Genéricos

- Considere o método printVetor da classe abaixo

```java
public class Generics {

    public static void main(String[] args) {

        double[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
        System.out.println("Vetor de double: ");
        printVetor(doubleArray);

        int[] intArray = {1, 2, 3, 4, 5, 6};
        System.out.println("Vetor de inteiros: ");
        printVetor(intArray);
    }

    public static void printVetor(double v[]) {
        for (double e : v) {
            System.out.println(e + " ");
        }
    }
}
```

- Qual seria o output ?
  - Erro de compilação no printVetor(arrayInt);
  - Embora o tipo **double** contenha o tipo **int**, uma referência para **double** não pode referenciar um vector de **int**
  - Seria necessário ter 2 implementações de printVetor

- O problema agravaria-se, à medida em que fosse necessário imprimir vetores de outros tipos de dados
  - **Para cada tipo, uma nova implementação**

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Métodos Genéricos


```
public class Generics {

    public static void main(String[] args) {

        double[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
        System.out.println("Vetor de double: ");
        printVetor(doubleArray);

        int[] intArray = {1, 2, 3, 4, 5, 6};
        System.out.println("Vetor de inteiros: ");
        printVetor(intArray);

        int[] charArray = {'E', 'C', 'D', 'A', '3', '0'};
        System.out.println("Vetor de char: ");
        printVetor(charArray);
    }

    public static void printVetor(double v[]) {
        for (double e : v) {
            System.out.println(e + " ");
        }
    }
}
```

Métodos Genéricos

- O problema anterior podia ser resolvido facilmente com um **método genérico**

```
public class Generics {

    public static void main(String[] args) {

        Dopuble[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
        System.out.println("Vetor de double: ");
        printVetor(doubleArray);

        Integer[] intArray = {1, 2, 3, 4, 5, 6};
        System.out.println("Vetor de inteiros: ");
        printVetor(intArray);

        Character[] charArray = {'E', 'C', 'D', 'A', '3', '0'};
        System.out.println("Vetor de char: ");
        printVetor(charArray);
    }

    public static<T> void printVetor(T v[]) {
        for (T e : v) {
            System.out.println(e + " ");
        }
    }
}
```

Metódo Genérico

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Métodos Genéricos


```java
public class Generics {

    public static void main(String[] args) {
        print("Generics");
        print(1904);
        print(new Gato("Pantufas", 2));
    }

    public static <T> void print(T value) {
        System.out.println(value + " ");
    }
}
```

- O método print é um método genérico capaz de imprimir qualquer coisa que receba como parâmetro.

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Métodos Genéricos


```java
public class Generics {

    public static void main(String[] args) {
        print("Daniel", 2);
    }

    public static <T, V> void print(T value, V otherValue) {
        System.out.println(value + " " + otherValue);
    }
}
```

- O método print recebe 2 valores genéricos como parâmetro

```java
public class Generics {

    public static void main(String[] args) {
        print("Daniel", 2);
    }
    public static <T, V> T print(T value, V otherValue) {
        return value;
    }
}
```

- O método print recebe 2 valores genéricos como parâmetro e retorna um valor genérico

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Limitar o alcance de um Genérico


```java
public class Printer<T> {

    T valueToPrint;

    public Printer(T valueToPrint) {
        this.valueToPrint = valueToPrint;
    }

    public void print() {
        System.out.println(valueToPrint);
    }
}
```

- A classe **Printer** criada anteriormente é capaz de imprimir qualquer tipo de dados como vimos anteriormente
- Podemos limitar por exemplo o nossa classe de modo que trate apenas de classes que herdem de Animal por exemplo.
- **Como assim?**

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Limitar o alcance de um Genérico


```java
public class Printer<T extends Animal> {

    T valueToPrint;

    public Printer(T valueToPrint) {
        this.valueToPrint = valueToPrint;
    }

    public void print() {
        System.out.println(valueToPrint);
    }
}
```

- Desta forma a **classe Printer** vai apenas conseguir tratar de classes que derivem de **Animal**

```java
Printer<Double> doublePrinter = new Printer<>(23.0);
printer.print();
```

Qual seria o Output ?

Erro de compilação

```java
Printer<Gato> printer = new Printer<>(new Gato("Pantufas", 2));
printer.print();
```

Qual seria o Output ?

Pantufas 2

- Desta forma o **T** será obrigatoriamente do tipo **Animal** logo **valueToPrint** vai ser capaz de aceder o que se encontra na classe **Animal**

valueToPrint.eat()

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Object vs Generic

- Qual a diferença entre utilizarmos o object e Generic?

```
ArrayList<Gato> gatos = new ArrayList<>();
gatos.add(new Gato("Pantufas", 2));
```

- O arraylist gatos apenas aceita gatos, se adiciona-se um cao ao mesmo obteria um erro

```
gatos.add(new Cao("Boby", 3));
```

- Antes dos generics criaríamos um ArrayList de **object** e assim já poderíamos adicionar ao array qualquer coisa.
  - O nosso código até funcionaria mas não seria 100% **"Type Safe".**

Autor: Sediangani Sofrimento
Ano Académico: 2025/2026

Object vs Generic


```
ArrayList<Object> gatos = new ArrayList<>();
gatos.add(new Gato("Pantufas", 2));

Gato gato  = (Gato) gatos.get(0);
```

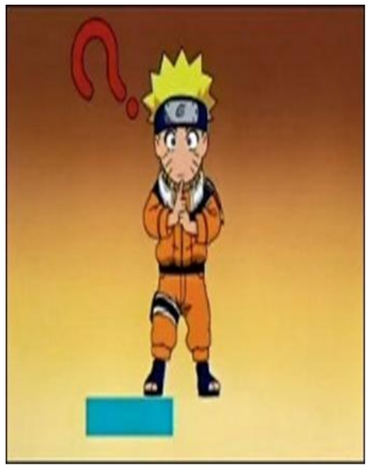
- Neste cenário não teríamos nenhuma excepção.

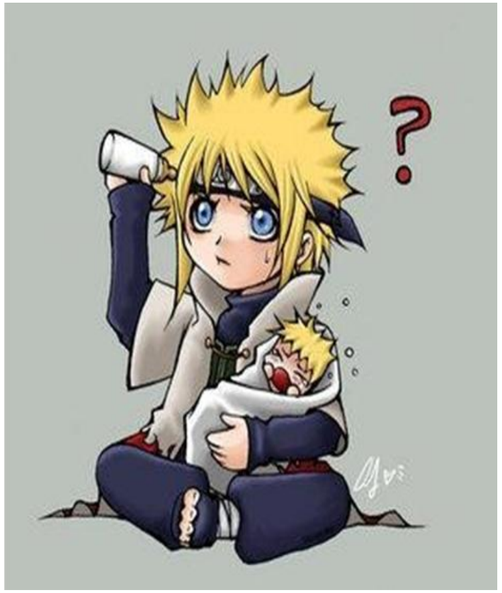
```
ArrayList<Object> gatos = new ArrayList<>();
gatos.add(new Cao("Boby", 3));

Gato gato  = (Gato) gatos.get(0);
```

- Neste cenário vamos ter uma excepção no CAST.

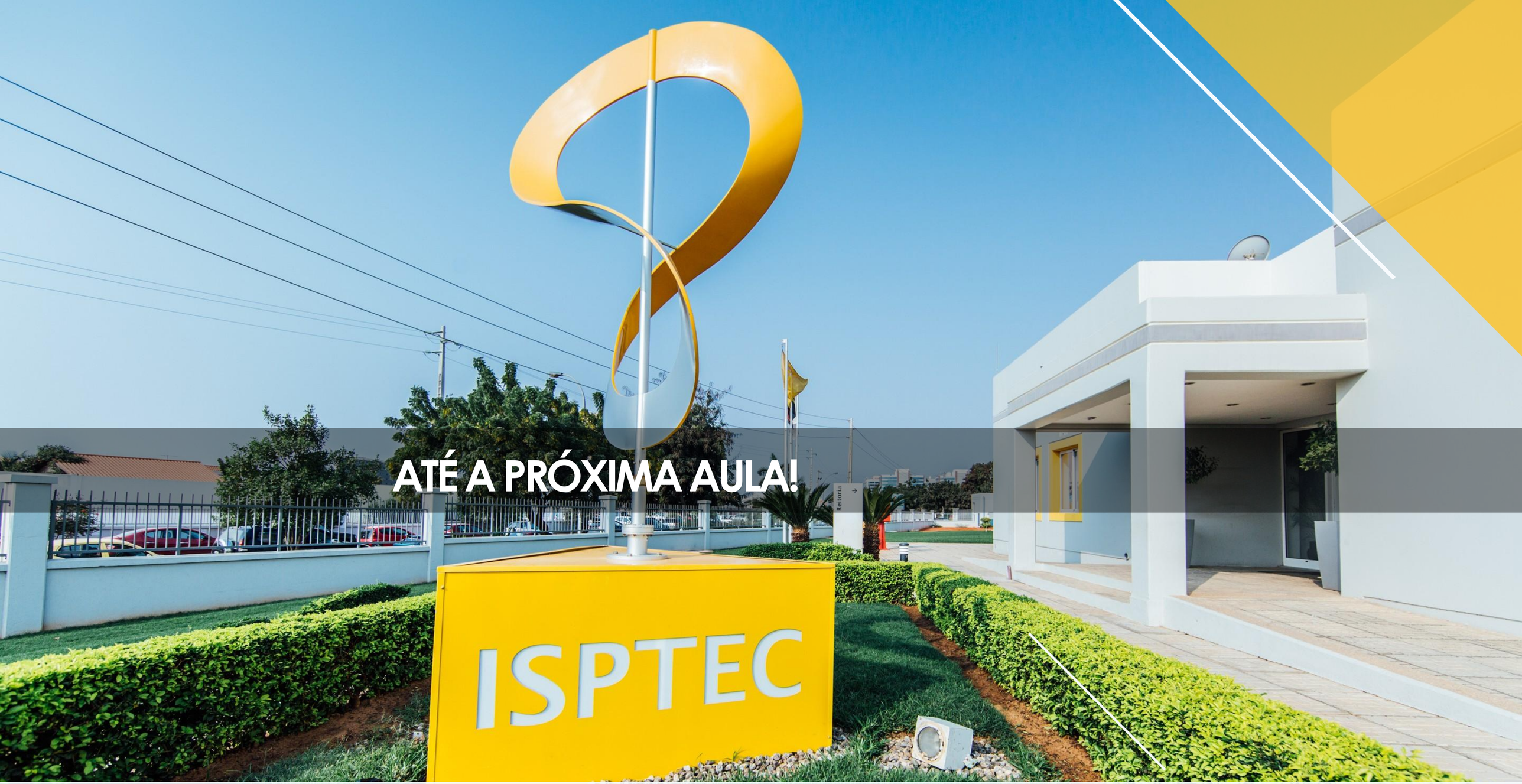
- Os **Generics** resolvem esse problema porque com eles podemos passar o tipo especifico que vai ser tratado sendo **100% "Type Safe".**

Dúvidas





Autor: Sediangani Sofrimento
Ano Académico: 2025/2026



ATÉ A PRÓXIMA AULA!