Curso de Java 8 para Web

Professor
Antonio Benedito Coimbra Sampaio Jr

abc | Treinamentos

www.abctreinamentos.com.br

Segunda

Disciplina

JAVA 8 - Pacotes, Tratamento de Exceções, Applets, Genéricos, Collections, Lambdas, Streams e Interfaces Gráficas

- **UNIDADE 1:** Pacotes, Erros e Exceções
- **UNIDADE 2:** Applets, Anotações e Entrada/Saída
- **UNIDADE 3: Genéricos**
- **UNIDADE 4:** Framework Collections
- **UNIDADE 5:** Novidades Java 8
- **UNIDADE 6:** Aplicações Gráficas em Java

UNIDADE 3

GENÉRICOS

Conceito de Genéricos

Genéricos

Definição:

- São tipos parametrizados que possibilitam a criação de classes, interfaces e métodos que funcionam automaticamente com tipos diferentes de dados.

- Com genéricos todos os *casts* são automáticos e implícitos, bem como a checagem de tipo é mais robusta em tempo de compilação.

- Foi uma importante funcionalidade que surgiu no Java 5 e trouxe duas significativas mudanças: **Sintaxe** e **reescrita do framework Collections**.

- O código escrito com genéricos não pode ser compilado pelas versões do Java anteriores à 5.

Genéricos

Código Sem o Uso de Genéricos:

```
List myIntList = new LinkedList();
myIntList.add(new Integer(0));
Integer x = (Integer) myIntList.iterator().next();
// O TYPE CAST (Integer) introduz a possibilidade de
// ERRO EM TEMPO DE EXECUÇÃO
Estacao x = (Estacao) myIntList.iterator().next();
// Exception in thread "main"
//java.lang.ClassCastException: java.lang.Integer
```

Código Com o Uso de Genéricos:

```
List<Integer> myIntList = new LinkedList<Integer>();
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

Genéricos

- Com o uso de Genéricos há maior 'estabilidade' no código escrito, fazendo com que mais erros sejam detectados em tempo de compilação, solução menos pior do que ser detectado em tempo de execução.

- Uma declaração de tipo genérico é compilada e gerada uma classe Java comum, como outra qualquer. Não há geração especial de código.

- Não aumenta o tamanho *bytecode* gerado.

- Genéricos funcionam como estruturas que aceitam parâmetros em tempo de execução.

Genéricos

Sintaxe Padrão

```
class NomeClasse <lista_parâmetros>
// Definição da Classe com o(s) Tipo(s)
//Genérico(s)
{...}

//Para instanciar uma classe Genérica
Box<Integer> integerBox = new Box<Integer>();
```

- A representação dos parâmetros em Genéricos segue a notação de apenas uma letra (em maiúscula): **T** de tipo; **V** de valor; **E** de Elemento; e **K** de *key* (chave).

Genéricos

Diamante

- A partir do Java 7, não é mais necessário informar os tipos de argumentos na instanciação de uma classe genérica, pois o compilador infere quais são os tipos de argumentos a partir do contexto.

- '**<>**' é informalmente chamada de **diamante**.

```
//Para instanciar uma classe Genérica com
//diamante
Box<Integer> integerBox = new Box<>();
```

Classe Genérica

```
public class ExemploGenerics <T>
{
   T ob;
   ExemploGenerics(T ob){
    this.ob = ob;
   }
   T getOb(){
    return this.ob;
   }
   void showType(){
     System.out.println("Tipo T e->"+
                        ob.getClass().getName());
   }
}
```

Uso da Classe Genérica

```
public class UsoGenerico {
  public static void main(String[] args)
  {
    ExemploGenerics<Integer> iob;
    iob = new ExemploGenerics<Integer>(88);
    iob.showType();
    int v = iob.getOb();
    System.out.println("Valor de v:"+v);
    ExemploGenerics<String> sob;
    sob = new ExemploGenerics<String>("Gen.");
    sob.showType();
    String s = sob.getOb();
    System.out.println("Valor de s:"+s);
  }
}
```

Exercícios

- 1) Implementar a classe **ExemploGenerics**.
- 2) Implementar a classe **UsoGenerico**.
- 3) Responda se o objeto **iob** é igual ao objeto **sob**?

```
iob = sob;(?)
```

Anatomia dos Genéricos

Anatomia da Classe Genérica

```
public class ExemploGenerics <T>
// T é o nome do Tipo do Parâmetro
```

```
T ob;
// ob é do tipo T que será especificado
quando o objeto ExemploGenerics for criado.
```

```
ExemploGenerics<Integer> iob;
iob = new ExemploGenerics<Integer>(88);
// T foi definido como Integer com valor 88.
```

```
ExemploGenerics<String> sob;
sob = new ExemploGenerics<String>("Teste");
// T foi definido como String "Teste".
int v = iob.getOb();
// AUTOBOXING
```

Anatomia da Classe Genérica

- É importante ressaltar que o Java cria apenas uma versão do objeto **ExemploGenerics**. Inicialmente ele é **Integer**. Posteriormente, todas as informações 'genéricas' são removidas e substituídas pelo tipo **String**.

- Um aspecto importante a ser observado é que a referência a uma versão de um tipo genérico não é compatível com a outra versão do mesmo genérico.

```
iob = sob; // ERRADO
```

- Genéricos previnem **erros em tempo de execução**.

RAW TYPE

- A tecnologia dos Genéricos foi adicionada no Java 5. Para evitar problemas de incompatibilidade com as versões anteriores do Java, todo código com Genéricos pode ser utilizado por aplicações sem essa funcionalidade.

- Nessa situação, uma classe Genérica utilizada sem qualquer argumento é conhecida como classe do tipo **RAW TYPE**.

```
ExemploGenerics ex = new ExemploGenerics();
// Necessária a definição do construtor vazio
// Criação de um RAW TYPE de ExemploGenerics
```

Genéricos com 02 Parâmetros

- Mais de um parâmetro pode ser declarado em um tipo Genérico.

```
public class ExemploGenerics2 <T,V>{
    T ob1;
    V ob2;
    ExemploGenerics2(T ob1, V ob2){
      this.ob1 = ob1;
      this.ob2 = ob2;
    }
    T getOb1(){
      return this.ob1;
    }
    V getOb2(){
      return this.ob2;
    }...
```

Genéricos com 02 Parâmetros

Exemplo de Uso:

```
public class UsoGenerico2 {
   public static void main(String[] args) {
     ExemploGenerics2<Integer, String> iob;
     iob = new ExemploGenerics2
                    <Integer,String>(88,"ABC");
     iob.showType();
     int v = iob.getOb1();
     String s = iob.getOb2();
     System.out.println("Valor de v:"+v);
     System.out.println("Valor de s:"+s);
   }
}
```

BOUNDED TYPES

- Em muitas situações é desejável a restrição do tipo de parâmetro “genérico” a ser utilizado.

```
public class Stats<T> {
   T[] num;
   Stats(T[] num){
      this.num = num;
   }
   double media(){
     double soma = 0;
     int i;
     for(i=0; i < num.length; i++){
       soma = soma + num[i].doubleValue(); // ERRO
     }
     return (soma/i);
   }
}
```

BOUNDED TYPES

- Com o recurso de ***Bounded Types*** é possível definir a qual superclasse o tipo genérico estende.

```
public class Stats<T extends Number> {
    T[] num;
    Stats(T[] num){
     this.num = num;
    }
    double media(){
      double soma = 0;
      int i;
      for(i=0; i < num.length; i++){
        soma = soma + num[i].doubleValue(); // OK
      }
      return (soma/i);
    }
}
```

BOUNDED TYPES

Exemplo de Uso:

```
...
public static void main(String[] args)
{
    Integer num[] = {1,2,3,4};
    Stats<Integer> stats = new Stats<Integer>(num);
    System.out.println(stats.media());
}
...
```

Exercícios

- 1) Implementar a classe **ExemploGenerics2**.
- 2) Implementar a classe **UsoGenerico2**.
- 3) Implementar a classe **Stats**.
- 4) Os trechos de código abaixo listados compilam?
  - A)
```
Set<String> strSet=new HashSet<String>();
strSet.add(new StringBuilder("hello"));
```
  - B)
```
Set<? extends Float> s = new TreeSet<Float>();
```
  - C)
```
LinkedList<int> list = new LinkedList<int>();
```

Métodos, Construtores e Interfaces Genéricos

Métodos Genéricos

- São aqueles métodos que utilizam argumentos de tipos genéricos em sua definição.

```
// Definição de um método genérico
<G extends Number>int intmedia(G num)
{
   return (num.intValue());
}
// Chamada do método genérico
Classe.<Integer>intmedia(5);
// ou
Classe.intmedia(5);
```

- É importante ressaltar que Métodos Genéricos podem ser definidos em classes “comuns” que não fazem uso de tipos genéricos e vice-versa.

Construtores Genéricos

- É possível a utilização de construtores com tipos genéricos em classes não “genéricas”.

```
public class NotaAluno {
    double valor;
    <T extends Number> NotaAluno(T obj)
    {
        valor = obj.doubleValue();
    }
}
```

Interfaces Genéricas

- Além das classes e métodos genéricos, também é possível criar **Interfaces Genéricas**.

```
interface MinMax<T extends Comparable>
{
    T min();
    T max();
}
```

- Toda classe que implementa uma interface genérica poderá ser ou não genérica.

```
class Teste <T extends Comparable>
                              implements MinMax {
   public T min(){return null;}
   public T max(){return null;}
}
```

Interfaces Genéricas

```
class Teste implements MinMax <Integer>
{
   public Integer min(){return null;}
   public Integer max(){return null;}
}
```

- São duas as principais vantagens em se utilizar as **Interfaces Genéricas**:
  1) Podem ser implementadas para diferentes tipos de dados;
  2) Limitam os tipos de dados que podem ser manipulados via Interface.

Exercícios

- 1) Criar um método e um construtor genérico na classe **NotaAluno**.

- 2) Implementar a interface genérica **MinMax**.

Hierarquia de Classes

Hierarquia de Classes

- Classes Genéricas também possuem as mesmas propriedades das Classes sem Genéricos para a construção de hierarquia de classes com superclasses e subclasses.

- A única diferença é que na hierarquia de Genéricos, qualquer tipo de argumento da superclasse deverá ser passado por todas as subclasses existentes.

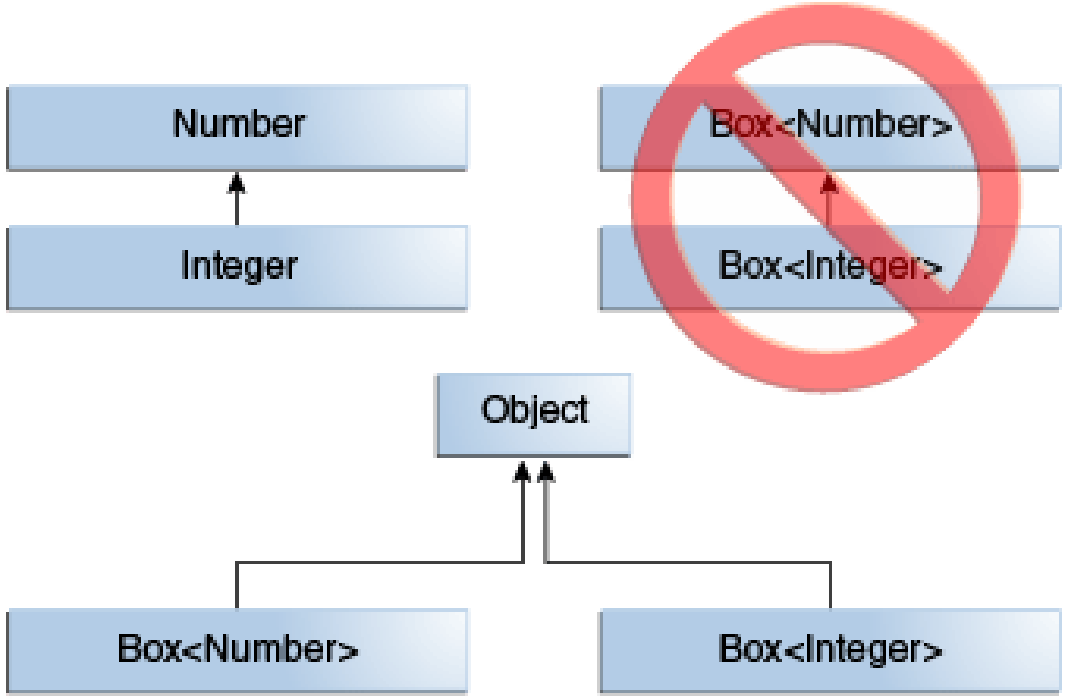
```
//SuperClasse Genérica
class Gen<T> {...}
//SubClasse Genérica
class Gen2<T> extends Gen<T> {...}
```

```
//SuperClasse Sem Genérico
class Gen {...}
//SubClasse Genérica
class Gen2<T> extends Gen {...}
```

Hierarquia de Classes

Se X é um subtipo de Y, e G um tipo genérico, **não** é verdade que G<X> é um subtipo de G<Y>

- Por exemplo, se uma classe Motorista é subclasse de Pessoa. Contudo, um tipo genérico de Motorista não é necessariamente subtipo do genérico Pessoa.



Hierarquia de Classes

Se X é um subtipo de Y, e G um tipo genérico, não é verdade que G<X> é um subtipo de G<Y>

```
Number numero;
Integer inteiro = new Integer(1);
numero = inteiro;
//OK
```

```
List<Number> number = new ArrayList<Number>();
List<Integer> integer = new ArrayList<Integer>();
number = integer;
//ERRO
```

Curingas (*Wildcards*)

- De acordo com a regra anterior:
  - **Collection<Object>** não é **super** tipo de **Collection<Integer>**
  - **Collection<Object>** só aceita componentes **Object**!

- O **super** tipo genérico verdadeiro é **Collection<?>**
  - **?** é o “tipo desconhecido”
  - **Collection<?>** é conhecido como ‘Coleção de objetos desconhecidos’

- A superclasse de qualquer Genérico é representada por **<?>**.

```
void printCollection(Collection<?> c) {
  for (Object e : c) {
    System.out.println(e);
  }
}
```

Bounded Wildcards

- Pode-se impor limites (superiores e inferiores) aos Curingas.

- **Limites Superiores <? extends T>**
  - Aceita T e todos os seus descendentes.
  - Por exemplo, se T for Collection, aceita List, Set, etc.

- **Limites Inferiores <? super T>**
  - Aceita T e todos os seus ascendentes.
  - Por exemplo, se T for ArrayList, aceita List, Collection, etc.

- Abaixo um exemplo de um método que faz uso de *Bounded Wildcards*.

```
public void drawAll(List<? extends Shape> shapes)
{ ... }
```

Exercícios

- 1) Os trechos de código abaixo listados compilam?
  - A)

```
class Shape { }
class Circle extends Shape { }
class Rectangle extends Shape { }
class Generics15 {
        public static void main(String[ ] args) {
            Vector<Shape> picture = new Vector<Shape>(); // 1
            picture.add(new Circle()); // 2
            picture.add(new Rectangle()); // 3
            Rectangle rect = picture.get(1); // 4
        }
}
```

  - B)

```
List<String> ls = new ArrayList<String>();
List<?> l = ls;
```

Restrições no Uso de Genéricos

Instanceof e Cast

- Uma classe não é mais sinônimo de tipo.
- Uma classe genérica pode ter muitos tipos.
- Não faz sentido o uso de **instanceof** nem de **cast**.

Instanceof e Cast

```
List<String> l1 = new ArrayList<String>();
List<Integer> l2 = new ArrayList<Integer>();
System.out.println(l1.getClass() == l2.getClass());
```

- O código acima imprime '**true**' porque todos os objetos de uma classe Genérica possuem a mesma classe em tempo de execução, independente do tipo de parâmetro utilizado.

```
List <String> string = new ArrayList<String>();
List <Integer> integer = new ArrayList<Integer>();
System.out.println(string.getClass().getName());
System.out.println(integer.getClass().getName());
System.out.println(string.getClass() ==
                                integer.getClass());
//java.util.ArrayList //java.util.ArrayList
//true
```

Vetor (Array)

- Vetores não podem ser utilizados com tipos parametrizados, apenas *Unbounded Wildcards*.

- Isto ocorre para manter o código seguro (‘*type safety*’).

```
List<String>[] lsa = new List<String>[10];
// ERRADO
List<?>[] lsb = new List<?>[10];
// CERTO
```

Outras Restrições no Uso de Genéricos

- Genéricos trabalham apenas com objetos, não sendo possível o uso de tipos primitivos.

```
ExemploGenerics<int> iob =
                        new ExemploGenerics<int>(88);
// ERRADO.
```

- Não é possível a criação de objetos dos tipos parametrizados

```
public static <E> void append(List<E> list) {
    E elem = new E();  // ERRO
    list.add(elem);
}
```

Outras Restrições no Uso de Genéricos

- Não é possível usar ***catch*** e ***throw*** para objetos de tipos parametrizados

```
class MathException<T> extends Exception
{...}    // ERRO
```

```
public static <T extends Exception, J>
                void execute(List<J> jobs) {
    try {
        for (J job : jobs)
            // ...
    } catch (T e) {// ERRO
        // ... }
}
```

Código com Genéricos e sem Genéricos

- Integrar código que faz uso intensivo de Genéricos com outro código que não faz uso, é mais comum do que o imaginado!
- Isto é devido ao fato de haver muito código legado escrito antes do Java 5 e a real necessidade de integração com código escrito depois do surgimento dessa importante melhoria do Java.
- É importante ressaltar os problemas decorrentes da utilização de código genérico e legado (sem genérico) na mesma aplicação afeta todas as garantias de segurança (código 'type safety') oferecidas pelo código genérico.

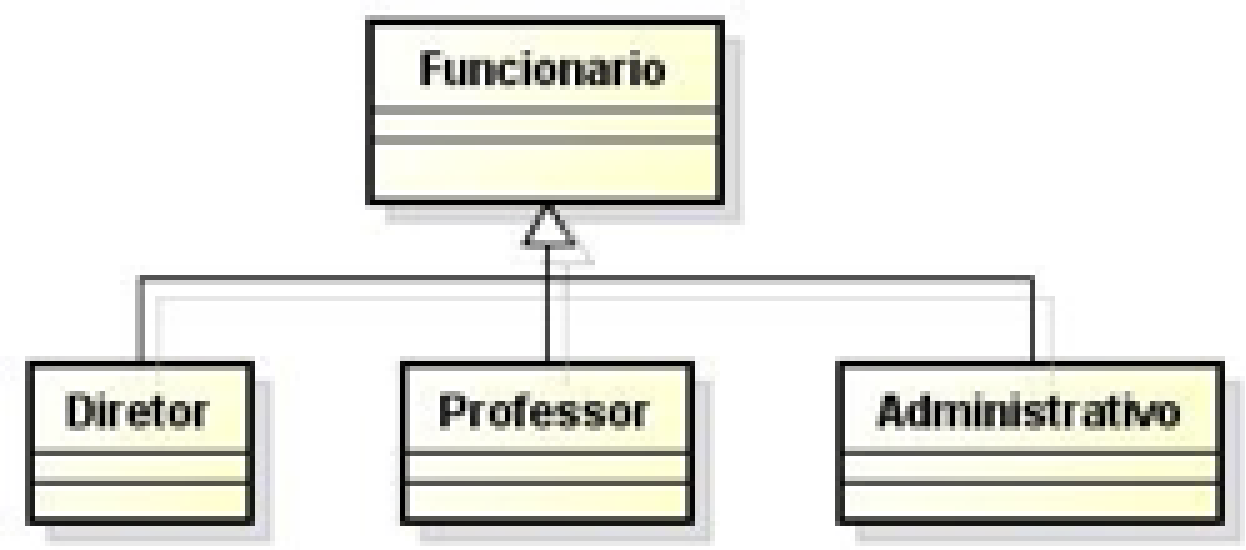
Código com Genéricos e sem Genéricos

Exemplo de Uso:

```
import java.util.*;
public class Inventory {
  public static Collection getParts(){
        return new ArrayList();
  }
  public static void main(String args[]){
    addAssembly("abc", new ArrayList());
    Collection<String> c = new ArrayList<>();
    Collection<Integer> k = getParts();
    //unchecked warnings!!!
  }
}
```

Exercícios

- 1) Criar o pacote **unidade3.persistencia**.
- 2) Criar a interface genérica IDAO<T>.
- 3) Criar a classe GenericDAO que implementa IDAO<T>.
- 4) Simular as operações de CRUD em um banco de dados fictício, fazendo uso da hierarquia de classes abaixo.



br.abctreinamentos.rh

RESUMO

TÓPICOS APRESENTADOS

- Nesta aula nós estudamos:

  - **Conceito de Genéricos**
  - **Anatomia dos Genéricos**
  - **Métodos, Construtores e Interfaces Genéricos**
  - **Hierarquia de Classes**
  - **Restrições no Uso de Genéricos**

ATIVIDADES PARA SE APROFUNDAR

- **1) Escreva um método genérico para listar os elementos de uma coleção de números inteiros, aqueles que são ímpar.**

- **2) Escreva um método genérico (swap(...)) que troque a posição de dois elementos de um vetor.**

- **3) O código abaixo compila? Justifique?**

```
public final class Algorithm {
    public static T max(T x, T y) {
        return x > y ? x : y;
    }
}
```

ATIVIDADES PARA SE APROFUNDAR

- **4) O código abaixo compila? Justifique?**

```
public class Singleton<T> {

    public static T getInstance() {
        if (instance == null)
            instance = new Singleton<T>();

        return instance;
    }

    private static T instance = null;
}
```